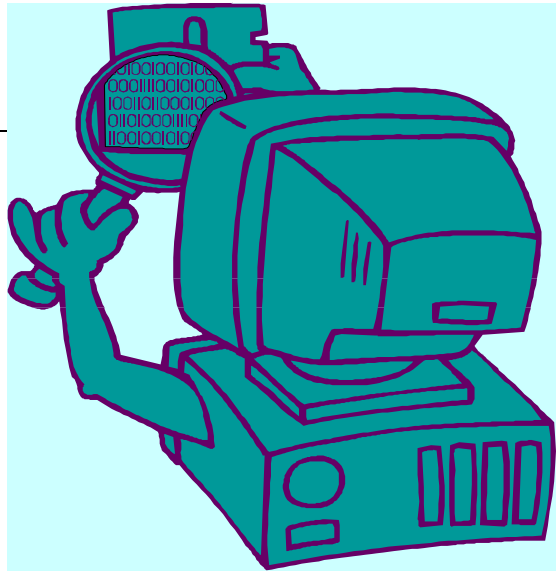


ภาษาโปรแกรม

(ภาษาซี)

1. ประวัติความเป็นมา

ภาษา		ภาษา		ภาษา
BCPL	➡	**B**	➡	**C**
Basic Combined Programming Language		บนเครื่อง **PDP-7 (UNIX)** พ.ศ. 2513		พ.ศ. 2515 โดย เดนนิช ริทชี่

2. โครงสร้างพื้นฐานของภาษาซี

```
# header                                  ส่วนที่ 1
main( )
{                /* เริ่มโปรแกรม */
    declaration            ส่วนที่ 2

     .........
     คำสั่งต่าง ๆ                          ส่วนที่ 3

}
```

ส่วนที่ 1 เป็นส่วนที่ระบุให้ซีคอมไพเลอร์เตรียมการทำงานที่กำหนดในส่วนนี้ไว้ โดยหน้าคำสั่งจะมีเครื่องหมาย # เช่น

```
# include <stdio.h>
```

เป็นการระบุให้นำไฟล์ stdio.h มารวมกับไฟล์นี้ เพื่อที่จะสามารถใช้คำสั่งที่อยู่ในไฟล์นี้มาใช้งานได้

หรือ

```
# define START 0
```

เป็นการกำหนดค่าคงที่ให้กับตัวแปร START โดยให้มีค่าเป็น 0

หรือ

```
# define temp 37
```

เป็นการกำหนดให้ตัวแปร temp มีค่าเท่ากับ 37

ส่วนที่ 2 declaration เป็นการกำหนดชนิดข้อมูลที่จะใช้ใน

โปรแกรมซึ่งตัวแปรหรือข้อมูลต่าง ๆ นั้นจะต้องถูกประกาศ(declare) ในส่วนนี้ก่อน จึงจะสามารถนำไปใช้ในโปรแกรมได้ เช่น

```
int stdno;
```

เป็นการกำหนดว่าตัวแปร stdno เป็นข้อมูลชนิดจำนวนเต็ม หรือ interger ซึ่งอาจได้แก่ค่า 0,4,-1,-3,…. เป็นต้น

```
float  score;
```

เป็นการกำหนดว่าตัวแปร score เป็นข้อมูลชนิดเลขมีจุดทศนิยม (floating point)ซึ่งอาจมีค่า 0.23, 1.34, -21.002, เป็นต้น

ส่วนที่ 3 Body คือส่วนของตัวโปรแกรม โดยจะต้องเริ่มต้นด้วยฟังก์ชัน main ( ) แล้วใส่เครื่องหมายกำหนดขอบเขตเริ่มต้นของตัวโปรแกรมคือ { หลังจากนั้นใส่คำสั่งหรือฟังก์ชันต่าง ๆ โดยแต่ละคำสั่งหรือฟังก์ชันนั้น ๆ จะต้องปิดด้วยเครื่องหมาย ; เมื่อต้องการจบโปรแกรมให้ใส่เครื่องหมาย } ปิดท้าย เช่น

```
main ( )
{           /* เริ่มต้นโปรแกรม */
      คำสั่งต่าง ๆ ;
      ฟังก์ชัน;
       ...............
       ...............
}           /* จบโปรแกรม */
```

เครื่องหมายต่าง ๆ

{ } - เป็นตัวกำหนดขอบเขตหรือบล็อกของฟังก์ชัน

( ) - เป็นการระบุตัวผ่านค่าหรืออาร์กิวเมนต์ให้กับฟังก์ชัน ถ้าภายในวงเล็บไม่มีข้อความใด ๆ แสดงว่าไม่มีตัวผ่านค่าที่ต้องการระบุสำหรับฟังก์ชันนั้น ๆ

/* */ - เป็นการกำหนด comment หรือข้อความ ที่ไม่ต้องการให้คอมไพเลอร์ปฏิบัติงาน ซึ่งข้อความที่อยู่ภายในเครื่องหมายนี้จะถือว่า <u>ไม่ใช่คำสั่งปฏิบัติงาน</u>

ตัวอย่างโปรแกรม

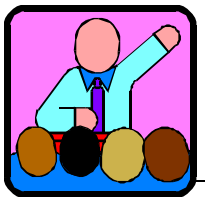
โปรแกรมที่ 1

```
# include <stdio.h>
int main (void )
{
    printf("Hello, Good morning. \n");
}
```

เป็นโปรแกรมสั่งพิมพ์ข้อความ Hello, Good morning.

โปรแกรมที่ 2

```
# include <stdio.h>
main ( )
{
        float   point;
        printf("\n\nPut your score in\n");
        scanf("%f", &point);
        printf("Your score is %f point\n\n", point);
}
```



เป็นโปรแกรมรับคะแนนและเก็บค่าที่ตัวแปร point หลังจากนั้นสั่งให้มีการพิมพ์คะแนนออกมา

ชนิดของข้อมูลและตัวแปรในภาษาซี

การกำหนดชื่อตัวแปร หลักการมีดังนี้

1. ต้องขึ้นต้นด้วยตัวอักษร
2. ห้ามใช้เครื่องหมายทางคณิตศาสตร์ในชื่อตัวแปร
3. สามารถใช้เครื่องหมาย underline '_' ได้
4. ห้ามใช้ reserved words เช่น int, float, etc.

Note: คอมไพเลอร์ในภาษาซีสามารถเห็นความแตกต่างของชื่อตัวแปรได้ยาวไม่เกิน 8 ตัวอักษร และชื่อตัวแปรจะแตกต่างกันถ้าใช้รูปแบบของตัวอักษรต่างกัน

แบบข้อมูลและขนาด

แบบข้อมูลหรือชนิดของตัวแปรต่าง ๆ ที่กำหนดไว้ในภาษาซี

char	ชนิดของตัวอักษรหรืออักขระ
int	ชนิดจำนวนเต็มปกติ
short	ชนิดจำนวนเต็มปกติ
long	ชนิดจำนวนเต็มที่มีความยาวเป็น 2 เท่า
unsigned	ชนิดของเลขที่ไม่คิดเครื่องหมาย
float	ชนิดเลขมีจุดทศนิยม
double	ชนิดเลขที่มีจุดทศนิยมความยาวเป็น 2 เท่า

ตารางแสดงเนื้อที่ในหน่วยความจำและค่าตัวเลขที่เก็บของข้อมูลแต่ละชนิด

ชนิดข้อมูล	เนื้อที่สำหรับเก็บ(ไบต์)	ค่าตัวเลขที่เก็บ
Char	1	เก็บตัวอักษร ASCII ได้ 1 ตัวหรือจำนวนเต็มระหว่าง 0 ถึง 255
Int	2	ค่าตัวเลขระหว่าง –32768 ถึง 32767
Short	2	ค่าตัวเลขระหว่าง –32768 ถึง 32767
Long	4	ค่าตัวเลขประมาณ ± 2000 ล้าน
Unsigned	Unsigned short = 2 Unsigned long = 4	ค่าตัวเลขระหว่าง 0 ถึง 65535 ค่าตัวเลขระหว่าง 0 ถึง 4000 ล้าน
Float	4	ได้ค่าตัวเลขยกกำลัง 10^x โดย x มีค่าระหว่าง –37 ถึง +38
Double	8	ความถูกต้องของตัวเลขจะมีค่าสูงขึ้น

ในการเขียนโปรแกรม แบบข้อมูลที่ใช้จะแบ่งออกเป็น 4 กลุ่มใหญ่ ดังนี้

- ข้อมูลและตัวแปรชนิดอักขระ
- ข้อมูลและตัวแปรชนิดจำนวนเต็ม
- ข้อมูลและตัวแปรชนิดเลขมีจุดทศนิยม
- ข้อมูลและตัวแปรแบบสตริง

ข้อมูลและตัวแปรชนิดอักขระ

1 อักขระแทนด้วย char โดยอยู่ภายในเครื่องหมาย ‘ ’ เช่น

```
# include <stdio.h>
main ( )
{
    char   reply;

    reply = ‘y’;
    ……................
}
```

การให้ค่าอักขระที่เป็นรหัสพิเศษหรือรหัสควบคุม

อักขระเหล่านี้ไม่สามารถให้ค่าโดยตรง แต่จะทำได้โดยการให้ค่าเป็นรหัส ASCII ซึ่งจะเขียนในรูปของเลขฐานแปด โดยใช้เครื่องหมาย '\' นำหน้า หรือใช้ตัวอักขระที่กำหนดให้กับรหัสนั้น ๆ เขียนตามเครื่องหมาย '\' สำหรับรหัสบางตัว เช่น

รหัส BELL แทนด้วย ASCII 007 ซึ่งกำหนดได้ดังนี้

beep = '\007';

หรือรหัสควบคุมการขึ้นบรรทัดใหม่ ตัวอักขระที่กำหนดให้กับรหัสคือ n

สามารถกำหนดเป็น newline = '\n';

ตัวอย่างโปรแกรม

```
# include <stdio.h>
main ( )
{
    char  newline;

     newline = '\n';
     printf("Hello, Good morning. %c",newline);
     printf("Hello, Good morning.\n");
}
```

ข้อมูลและตัวแปรชนิดจำนวนเต็ม

จำนวนเต็มในภาษาซีสามารถใช้แทนได้ 4 รูปแบบคือ int, short, long และ unsigned

สำหรับการกำหนดตัวแปรแบบ unsigned คือจำนวนเต็มที่ไม่คิดเครื่องหมายนั้นจะต้องใช้ควบคู่กับรูปแบบข้อมูลจำนวนเต็มชนิดอื่น ๆ คือ int หรือ short หรือ long ตัวอย่างเช่น

```
unsigned  int   plusnum;
unsigned long  width;
unsigned short absno;  /* absolute number */
```

ข้อมูลและตัวแปรชนิดเลขมีจุดทศนิยม

สำหรับเลขมีจุดทศนิยมนั้นแทนได้ 2 แบบคือ float และ double โดย double เก็บค่าได้เป็น 2 เท่าของ float สำหรับงานทางวิทยาศาสตร์ที่ต้องการความละเอียดในการเก็บค่า มักใช้การเก็บในรูปแบบนี้ คือเก็บแบบเอ็กโพเนนซ์ ดังตัวอย่างต่อไปนี้

ตัวเลข	แสดงแบบวิทยาศาสตร์	แบบเอ็กโพเนนซ์
9,000,000,000	$9.0*10^{9}$	9.0e9
345,000	$3.45*10^{5}$	3.45e5
0.00063	$6.3*10^{-4}$	6.3e-4
0.00000924	$9.24*10^{-6}$	9.24e-6

ข้อมูลและตัวแปรแบบสตริง

สตริงหมายถึงตัวอักขระหลาย ๆ ตัวมาประกอบกันเป็นข้อความ ซึ่งการที่นำตัวแปรหลาย ๆ ตัวมาเก็บรวมกันในภาษาซีนี้เรียกว่า อะเรย์ (array) ดังนั้นข้อมูลแบบสตริงคือ อะเรย์ของตัวอักขระ นั่นเอง

เครื่องหมายของอะเรย์คือ [ ] รูปแบบการกำหนดสตริงจึงมีลักษณะดังนี้

```
char    name[30];
```

หมายถึง ตัวแปร name เป็นชนิด char ที่มีความยาว 30 ตัวอักษร โดยเก็บเป็น อะเรย์ การเก็บนั้นจะเก็บเรียงกันทีละไบต์ และไบต์สุดท้ายเก็บรหัส null คือ \0 ดังนั้นจะเก็บได้จริงเพียง 29 ตัวอักษร

การกำหนดค่าให้ตัวแปรและการส่งผลลัพธ์

การกำหนดค่าให้ตัวแปรอาจทำได้โดยกำหนดในโปรแกรม หรือกำหนดในขณะที่มีการกำหนดชนิดก็ได้ เช่น

```
main ( )
{
    int age = 18;
    float height;

    height = 172.5;
    printf("Mr. Surasak is %d years old",age);
    printf(" and tall %f cms.\n",height);
}
```

ตัวอย่างของโปรแกรมในการกำหนดค่าและส่งค่าผลลัพธ์

```
# include <stdio.h>
main ( )
{
    int sum,valuea;
    int count = 1;

    valuea = 4;
    sum = count + valuea;
    printf("Total value is %d.\n",sum);
}
```

ผลลัพธ์จะปรากฏข้อความ : Total value is 5.

ฟังก์ชัน printf( ) และ scanf( )

รูปแบบของ printf ( )

printf(ส่วนควบคุมการพิมพ์, อาร์กิวเมนต์, อาร์กิวเมนต์,...)

ส่วนควบคุมการพิมพ์ เป็นสตริงที่มีข้อความและรูปแบบของการพิมพ์โดยอยู่ในเครื่องหมาย “ ”

อาร์กิวเมนต์ เป็นส่วนที่จะนำข้อมูลมาพิมพ์ ตามรูปแบบที่กำหนดมาในส่วนควบคุมการพิมพ์

รูปแบบที่ใช้สำหรับกำหนดการพิมพ์ในฟังก์ชัน printf

%d พิมพ์ด้วยเลขฐานสิบ

%o ” ” เลขฐานแปด

%x ” ” เลขฐานสิบหก

%u ” ” เลขฐานสิบแบบไม่คิดเครื่องหมาย

%e ” ” ตัวเลขแบบวิทยาศาสตร์ เช่น 2.13e45

%f ” ” ตัวเลขมีจุดทศนิยม

%g ” ” รูปแบบ %e หรือ %f โดยเลือกแบบที่สั้นที่สุด

สำหรับสตริงมีรูปแบบการพิมพ์ดังนี้

%c พิมพ์ด้วยตัวอักษรตัวเดียว

%s ” ”ข้อความ

เครื่องหมายสำหรับปรับเปลี่ยนรูปแบบของข้อมูล

เครื่องหมายลบ ให้พิมพ์ข้อมูลชิดขอบซ้าย
(ปกติข้อมูลทั้งหมดจะพิมพ์ชิดขวา)

สตริงตัวเลข ระบุความกว้างของฟิลด์

จุดทศนิยม เป็นการกำหนดความกว้างของจุดทศนิยม

Note การปรับเปลี่ยนรูปแบบของข้อมูลนี้ทำได้โดย การใส่เครื่องหมายเหล่านี้ระหว่างเครื่องหมาย % และเครื่องหมายที่กำหนดรูปแบบการพิมพ์

รูปแบบของ scanf ( )

scanf(ส่วนควบคุมข้อมูล, อาร์กิวเมนต์, อาร์กิวเมนต์,...)

ส่วนควบคุมข้อมูล เป็นการกำหนดรูปแบบข้อมูลในเครื่องหมาย “ ”

อาร์กิวเมนต์ เป็นส่วนที่จะนำข้อมูลมาเก็บ(ในตัวแปร) ซึ่งชนิดของข้อมูลต้องตรงตามรูปแบบที่กำหนดในส่วนควบคุมข้อมูล

การกำหนดลักษณะอาร์กิวเมนต์มีได้ 2 แบบดังนี้

ถ้าข้อมูลนั้นอาจจะนำไปใช้ในการคำนวณ

- จะใส่เครื่องหมาย & หน้าตัวแปร

ถ้าข้อมูลนั้นเป็นข้อความที่จะนำไปเก็บไว้ในตัวแปรเลย

- ไม่จำเป็นต้องใส่เครื่องหมาย & หน้าตัวแปร

โอเปอเรเตอร์และนิพจน์

การแทนโอเปอเรเตอร์ทางคณิตศาสตร์สำหรับภาษาซี

+	การบวก
-	การลบ
*	การคูณ
/	การหาร
%	การหารเอาเศษ (โมดูลัส)

การเปลี่ยนชนิดของข้อมูล

ทำได้โดยระบุชนิดที่ต้องการเปลี่ยนภายในเครื่องหมาย ( ) แล้ววางหน้าตัวแปรหรือข้อมูลที่ต้องการเปลี่ยนแปลงชนิด

```
float  money;
```

ต้องการเปลี่ยนตัวแปร float ไปเป็น integer ทำได้ดังนี้

```
    (int) money;
```

```
int cost;
      cost = 2.7+4.5;
      cost = (int)2.7+(int)4.5;
```

การเพิ่มค่าและลดค่าตัวแปร

++n เพิ่มค่า n อีก 1

- -n ลดค่า n ลง 1

ความแตกต่างระหว่าง count++ และ ++count

เช่น

count = 5;

x = count++; จะได้ค่า x เท่ากับ 5

แล้วค่า count เท่ากับ 6

count = 5;

x = ++count; จะได้ค่า x เท่ากับ 6

นิพจน์กำหนดค่า (Assignment expression)

เครื่องหมายที่ใช้กำหนดค่าคือ =

โดยเป็นการกำหนดค่าทางขวาของเครื่องหมาย ให้กับตัวแปรที่อยู่ทางซ้าย เช่น j = 7+2

หรือ k = k+4

3.4.6 เครื่องหมายและนิพจน์เปรียบเทียบ

>	หรือ >=	มากกว่า หรือ มากกว่าเท่ากับ
<	หรือ <=	น้อยกว่า หรือ น้อยกว่าเท่ากับ
	==	เท่ากับ
	!=	ไม่เท่ากับ

ความแตกต่างของเครื่องหมาย = และ ==

เครื่องหมาย = เป็นตัวกำหนดค่า

ในขณะที่เครื่องหมาย == เป็นเครื่องหมายเปรียบเทียบ ตัวอย่างเช่น

point = 44;

หมายถึง เป็นการกำหนดค่าให้กับตัวแปร point ให้มีค่าเท่ากับ 44

point == 44;

หมายถึง เป็นการตรวจสอบว่าค่า **point** มีค่าเท่ากับ 44 หรือไม่

เครื่องหมายและนิพจน์เปรียบเทียบแบบตรรกศาสตร์

&& และ (and)

|| หรือ (or)

! ไม่ (not)

ค่าของนิพจน์เปรียบเทียบเชิงตรรก

นิพจน์ที่ 1 && นิพจน์ที่ 2 เป็นจริง เมื่อนิพจน์ทั้งสองเป็นจริง

นิพจน์ที่ 1 || นิพจน์ที่ 2 เป็นจริง เมื่อนิพจน์ใดนิพจน์หนึ่ง เป็นจริงหรือ ทั้งสองนิพจน์นั้นเป็นจริง

! นิพจน์เปรียบเทียบ เป็นจริง เมื่อนิพจน์เปรียบเทียบเป็นเท็จ

คำสั่ง *if*

รูปแบบของคำสั่ง

if (เงื่อนไข)

คำสั่งที่ต้องทำ ถ้าเงื่อนไขนั้นเป็นจริง;

ตัวอย่างเช่น

```
if  (score >= 80)
    grade = 'A'; /* simple statement  */
```

หรือ

```
if (math >= 60 && eng >= 55)
{    grade = 'S';  /* compound statement */
     printf("Your grade is %c\n",grade);
 }
```

คำสั่ง if ….. else …..

รูปแบบของคำสั่ง

if (**คำสั่งหรือนิพจน์เงื่อนไข**)

คำสั่งที่ต้องทำเมื่อเงื่อนไขนั้นเป็นจริง

else **คำสั่งที่ต้องทำเมื่อเงื่อนไขนั้นไม่เป็นจริง**

ตัวอย่างเช่น

```
if (value1  > value2)
        min = value2;
else
         min = value1;
```

เครื่องหมายพิเศษที่ใช้ในการเปรียบเทียบเงื่อนไข ? :

รูปแบบทั่วไปของคำสั่งเปรียบเทียบเงื่อนไข ? : มีดังนี้

นิพจน์ที่ 1 ? นิพจน์ที่ 2 : นิพจน์ที่ 3

ความหมายคือ

if **นิพจน์ที่ 1 เป็นจริง**

ทำตามคำสั่งในนิพจน์ที่ 2

else

ทำตามคำสั่งในนิพจน์ที่ 3

เช่น x = (y< 0) ? -y : y;

คำสั่งตรวจสอบเงื่อนไขหลาย ๆ ทาง : switch และ break

รูปแบบคำสั่ง

```
switch (นิพจน์)
    {
        case  label1   : statement1;
          case  label2   : statement2;
        ................
        ................
        default        :  statementn;
    }
```

ตัวอย่าง

```
switch   (ch)
  {
        case '1'    :
              printf("Red\n");
        case '2'    :
              printf("Blue\n");
         case '3'    :
              printf("Yellow\n");
        default     :
              printf("White\n");
  }
```

ตัวอย่าง

```
switch   (ch)
{
        case '1' : printf("Red\n");
            break;
        case '2' : printf("Blue\n");
            break;
        case '3' : printf("Yellow\n");
                    break;
        default :  printf("White\n");
    }
```

คำสั่ง loop หรือคำสั่งวนซ้ำ

คำสั่งลูป while

รูปแบบ

```
while   (นิพจน์เงื่อนไข)
{
    คำสั่งที่วนลูป;
    ............
  .............
}
```

compound statements

คำสั่งลูป for

รูปแบบ

```
for ( นิพจน์ที่ 1  ;   นิพจน์ที่ 2 ;  นิพจน์ที่ 3 )
 {
      คำสั่งวนรอบ;
      .......
 }
```

เป็นคำสั่งที่ใช้ในการควบคุมให้มีการวนรอบคำสั่งหลาย ๆ รอบ โดยนิพจน์ที่ 1 คือการกำหนดค่าเริ่มต้นให้กับตัวแปรที่ใช้ในการวนรอบ นิพจน์ที่ 2 เป็นการเปรียบเทียบ ก่อนที่จะวนรอบถ้าเงื่อนไขของนิพจน์เป็นจริงจะมีการทำงานตามคำสั่งวนรอบ นิพจน์ที่ 3 เป็นคำสั่งในการกำหนดค่าที่จะเปลี่ยนแปลงไปในแต่ละรอบ

คำสั่งวนรอบแบบที่ตรวจสอบเงื่อนไขทีหลัง : do while

รูปแบบ

```
do
    statement;
while (นิพจน์เงื่อนไข);
```

เช่น

```
num = 2;
do
    {
        num++;
        printf("Now no is %d\n",num);
    } while (num == 10)
```

คำสั่งควบคุมอื่น ๆ **break, continue, goto** และ **labels**

คำสั่ง break

ใช้เมื่อต้องการให้การทำงานสามารถหลุดออกจากลูปและกระโดดไปยังคำสั่งที่อยู่นอกลูปทันที โดยไม่ต้องตรวจสอบเงื่อนไขใด ๆ

คำสั่ง continue

ใช้เมื่อต้องการให้การทำงานนั้น ย้อนกลับไปวนรอบใหม่อีกครั้ง ซึ่งมีลักษณะที่ตรงข้ามกับคำสั่ง break

คำสั่ง goto และ labels

คำสั่ง goto ประกอบด้วย 2 ส่วน คือ

- ตัวคำสั่ง goto เป็นคำสั่งให้กระโดดไปยังตำแหน่งที่กำหนด โดยจะกำหนดเป็นชื่อ เรียกว่า label name
- ชื่อ (label name) เป็นตัวกำหนดตำแหน่งที่คำสั่งจะกระโดดไปทำงาน

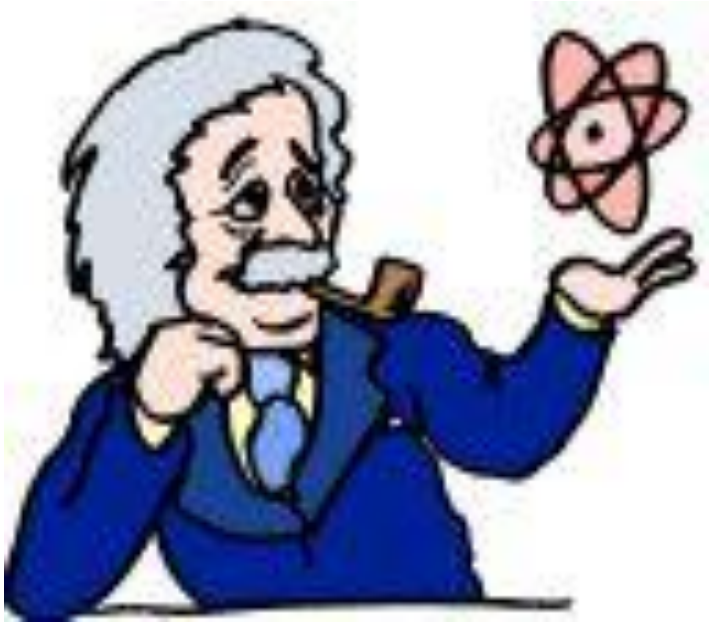
ข้อควรระวัง ! คำสั่งนี้ถือเป็นคำสั่งที่ควรหลีกเลี่ยงในการเขียนโปรแกรม แต่ถ้าจำเป็นหรือหลีกเลี่ยงไม่ได้เท่านั้น จึงจะใช้คำสั่งนี้

ตัวอย่างโปรแกรมที่ใช้คำสั่ง goto

```
#include<stdio.h>
main()
{    int sum,n;


     for(n=1;n<10;n++)
          if (n==5)
                    goto part1;
          else printf("%d\n",n);
     part1 :  printf("Interrupt with no. 5\n");
}
```

ฟังก์ชัน (Function)



ฟังก์ชัน (Functions)

การออกแบบโปรแกรมในภาษาซีจะอยู่บนพื้นฐานของการออกแบบโมดูล (Module Design) โดยการแบ่งโปรแกรมออกเป็นงานย่อย ๆ (หรือโมดูล) แต่ละงานย่อยจะทำงานอย่างใดอย่างหนึ่งเท่านั้น และไม่ควรจะมีขนาดใหญ่จนเกินไป งานย่อยเหล่านี้เมื่อนำไปเขียนโปรแกรมในภาษาซีจะเป็นการเขียนในลักษณะของฟังก์ชัน

ตัวอย่าง

โปรแกรมเพื่อบวกเลขสองจำนวนที่รับจากผู้ใช้และแสดงผลการคำนวณ

สามารถแบ่งการทำงานเป็นงานย่อยได้ดังนี้

รับข้อมูล 2 จำนวนจากผู้ใช้

บวกเลข 2 จำนวนแล้วเก็บผลลัพธ์

แสดงผลลัพธ์ของการทำงาน

ตัวอย่าง (ต่อ)

จะได้ว่าโปรแกรมประกอบด้วยฟังก์ชัน 4 ฟังก์ชันคือ

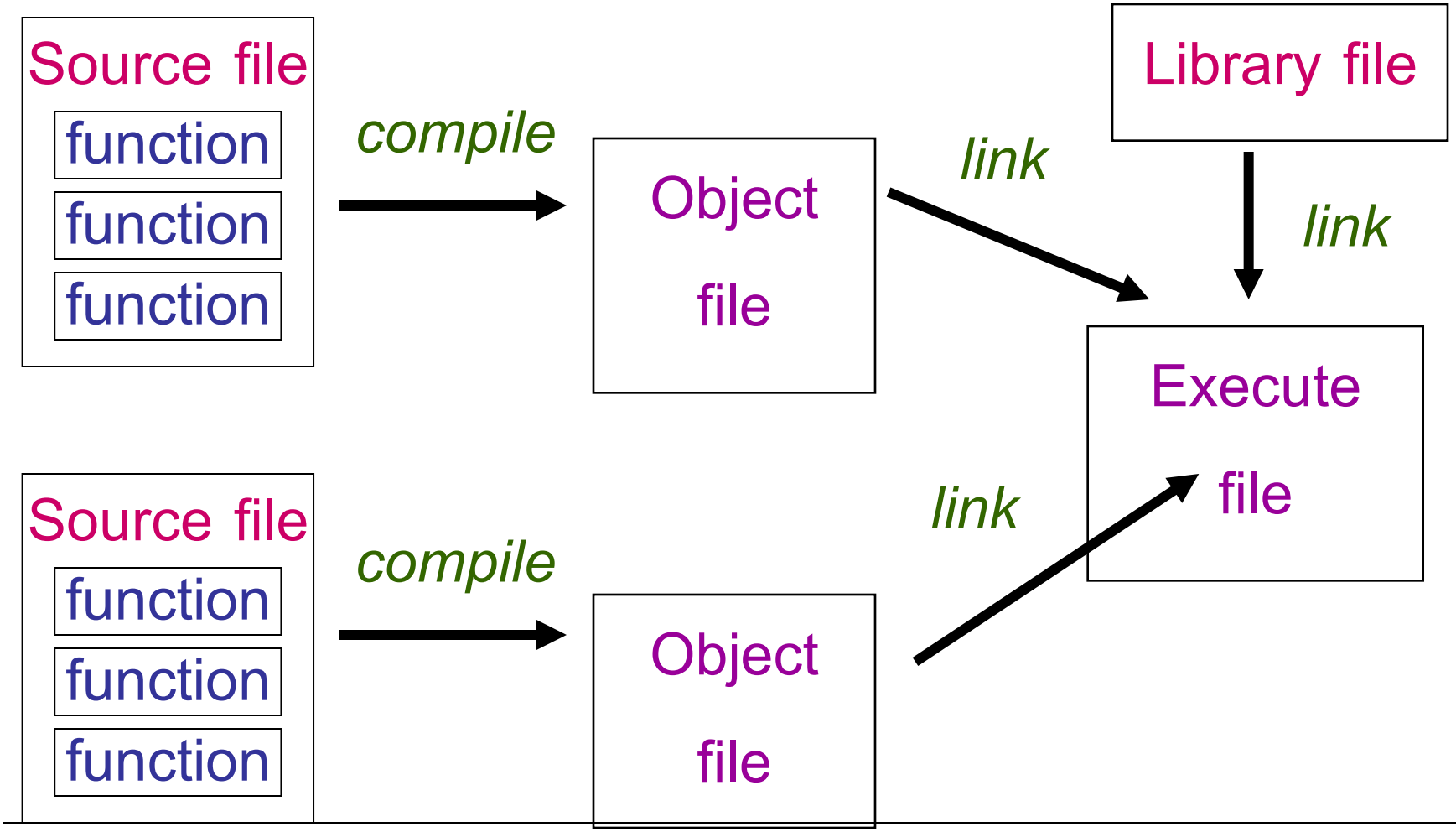
ฟังก์ชันหลัก

ฟังก์ชันการรับข้อมูล

ฟังก์ชันในการบวกเลข

ฟังก์ชันแสดงผลลัพธ์

ขั้นตอนการสร้างโปรแกรมด้วยภาษา C



4.1 รูปแบบของฟังก์ชัน

แบบที่ 1

int , char , float , double ฯลฯ

ชนิดข้อมูลที่คืนค่า ชื่อฟังก์ชัน (การประกาศตัวแปร)

```
{
        การประกาศตัวแปรภายในฟังก์ชัน;
          คำสั่ง;
          return  (ค่าข้อมูลที่ต้องการส่งค่ากลับ);
}
```

รูปแบบของฟังก์ชัน (ต่อ)

แบบที่ 2

```
void   ชื่อฟังก์ชัน ( การประกาศตัวแปร )
{
        การประกาศตัวแปรภายในฟังก์ชัน;
        คำสั่ง;
}
```

ตัวอย่าง 4.1

แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้

```
#include <stdio.h>
double  InputDouble ( )
{
        double  x;
        printf ( “\nInput real value : “ );
        scanf ( “%.2f  ”, &x );
        return ( x );
}
```

ตัวอย่าง 4.1 (ต่อ)

```
double  SumDouble ( double  x,  double  y )
{
        return ( x + y );
}
void  PrintOut ( double  x )
{
        printf ( “\n Result of sum is : %.2f”,  x );
}
```

ตัวอย่าง 4.1 (ต่อ)

```
void  main (  )
{
        double   a1, a2, sumVal;
        a1  =  InputDouble( );
        a2  =  InputDouble( );
        sumVal  =  SumDouble ( a1, a2 );
        PrintOut ( sumVal );
}
```

4.2 การประกาศโพรโทไทป์ของฟังก์ชัน

การประกาศโปรโตไทป์เป็นสิ่งจำเป็นในภาษาซีเนื่องจากภาษาซีเป็นภาษาในลักษณะที่ต้องมีการประกาศฟังก์ชันก่อนจะเรียกใช้ฟังก์ชันนั้น (Predefined Function)

จากตัวอย่างที่ 4.1 จะเห็นว่าฟังก์ชัน main ( ) จะอยู่ใต้ฟังก์ชันอื่น ๆ ที่มีการเรียกใช้ เป็นลักษณะที่ต้องประกาศฟังก์ชันที่ต้องการเรียกใช้ก่อนจากเรียกใช้ฟังก์ชันนั้น แต่หากต้องการย้ายฟังก์ชัน main ( ) ขึ้นไปไว้ด้านบน จะต้องมีการประกาศโปรโตไทป์ของฟังก์ชันที่ต้องการเรียกใช้ก่อนเสมอ

ตัวอย่าง 4.2

แสดงการทำงานของโปรแกรมการบวกเลขจำนวนจริง 2 จำนวนที่รับจากผู้ใช้ในลักษณะที่มีการประกาศโปรโตไทป์

```
#include <stdio.h>
double  InputDouble ( );
double  SumDouble ( double ,  double  );
void     PrintOut ( double );
```

ตัวอย่าง 4.2 (ต่อ)

```
void  main (  )
{
        double   a1, a2, sumVal;
        a1  =  InputDouble( );
        a2  =  InputDouble( );
        sumVal  =  SumDouble ( a1, a2 );
        PrintOut ( sumVal );
}
```

จะเห็นว่าในโปรโตไทป์ไม่มีการประกาศชื่อตัวแปร มีแต่การเขียนประเภทของตัวแปรไว้ภายในเป็นการช่วยให้คอมไพเลอร์สามารถตรวจสอบจำนวนของตัวแปร ประเภทของตัวแปร ประเภทของการคืนค่า ภายในโปรแกรมว่ามีการเรียกใช้งานสิ่งต่าง ๆ เกี่ยวกับฟังก์ชันนั้นถูกต้องหรือไม่ นอกจากนี้เราอาจจะแยกส่วนโปรโตไทป์ไปเขียนไว้ในอินคลูชไฟล์ก็ได้เช่นเดียวกัน

4.3 การเรียกใช้ฟังก์ชัน

การเรียกใช้ฟังก์ชันที่มีการคืนค่า จะใช้รูปแบบดังต่อไปนี้

ค่าที่รับ = ฟังก์ชัน (อาร์กิวเมนต์)

ตัวอย่าง

a1 ต้องมีชนิดเป็น double เนื่องจากค่าที่จะส่งคืนกลับมาจากฟังก์ชันมีชนิดเป็น double

```
a1 = InputDouble ( );
```

ใช้คู่กับโปรโตไทป์

```
double InputDouble ( );
```

ตัวอย่าง

a1 และ a2 ต้องมีชนิดเป็น double
เพื่อให้ตรงกับชนิดตัวแปรของอาร์กิวเมนท์
ที่ประกาศในโปรโตไทป์

```
sumVal =  SumDouble (a1,a2 );
```

ใช้คู่กับโปรโตไทป์

```
double  InputDouble ( );
```

ตัวอย่าง

```
PrintOut( sumVal );
```

ใช้คู่กับโปรโตไทป์

```
void PrintOut ( double );
```

ประกาศให้รู้ว่าฟังก์ชันนี้ไม่มีการคืนค่า

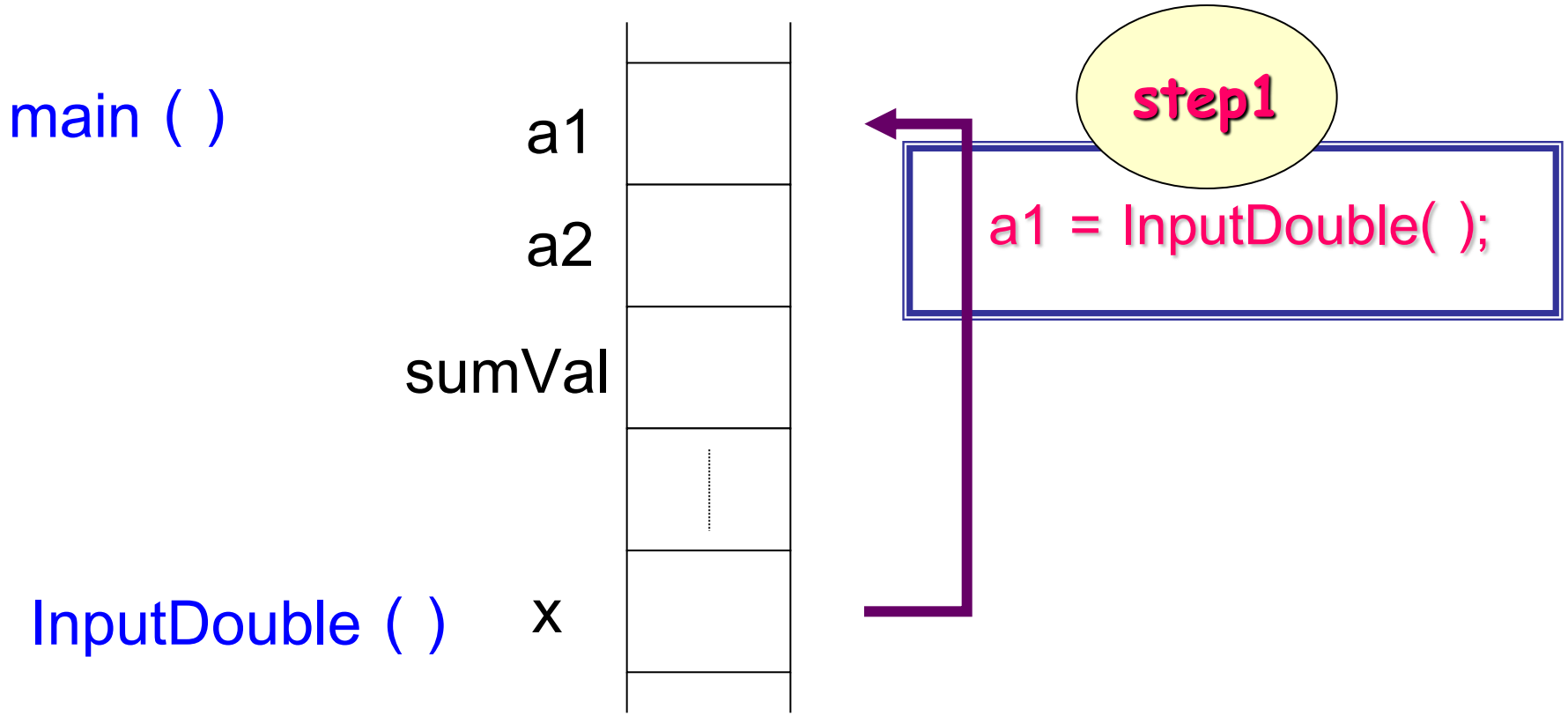
4.4 ขอบเขต (Scope)

การทำงานของโปรแกรมภาษาซีจะทำงานที่ฟังก์ชัน main ( ) ก่อนเสมอ เมื่อฟังก์ชัน main ( ) เรียกใช้งานฟังก์ชันอื่น ก็จะมีการส่งคอนโทรล (Control) ที่ควบคุมการทำงานไปยังฟังก์ชันนั้น ๆ จนกว่าจะจบฟังก์ชัน หรือพบคำสั่ง return

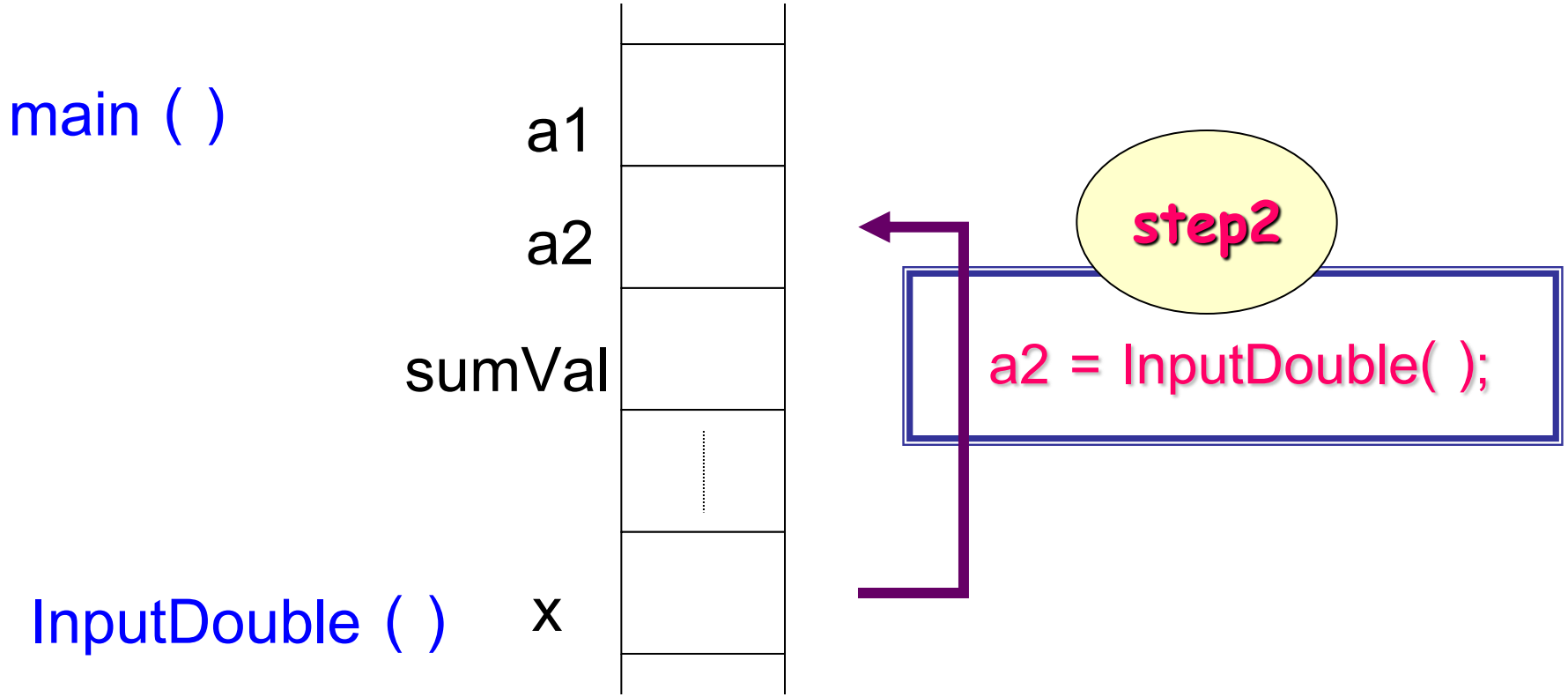
เมื่อมีการเรียกใช้งานฟังก์ชันจะมีการจองพื้นที่หน่วยความจำสำหรับตัวแปรที่ต้องใช้ภายในฟังก์ชันนั้น และเมื่อสิ้นสุดการทำงานของฟังก์ชันก็จะมีการคืนพื้นที่หน่วยความจำส่วนนั้นกลับสู่ระบบ การใช้งานตัวแปรแต่ละตัวจะมีขอบเขตของการใช้งานขึ้นอยู่กับตำแหน่งที่ประกาศตัวแปรนั้น

ตัวอย่าง

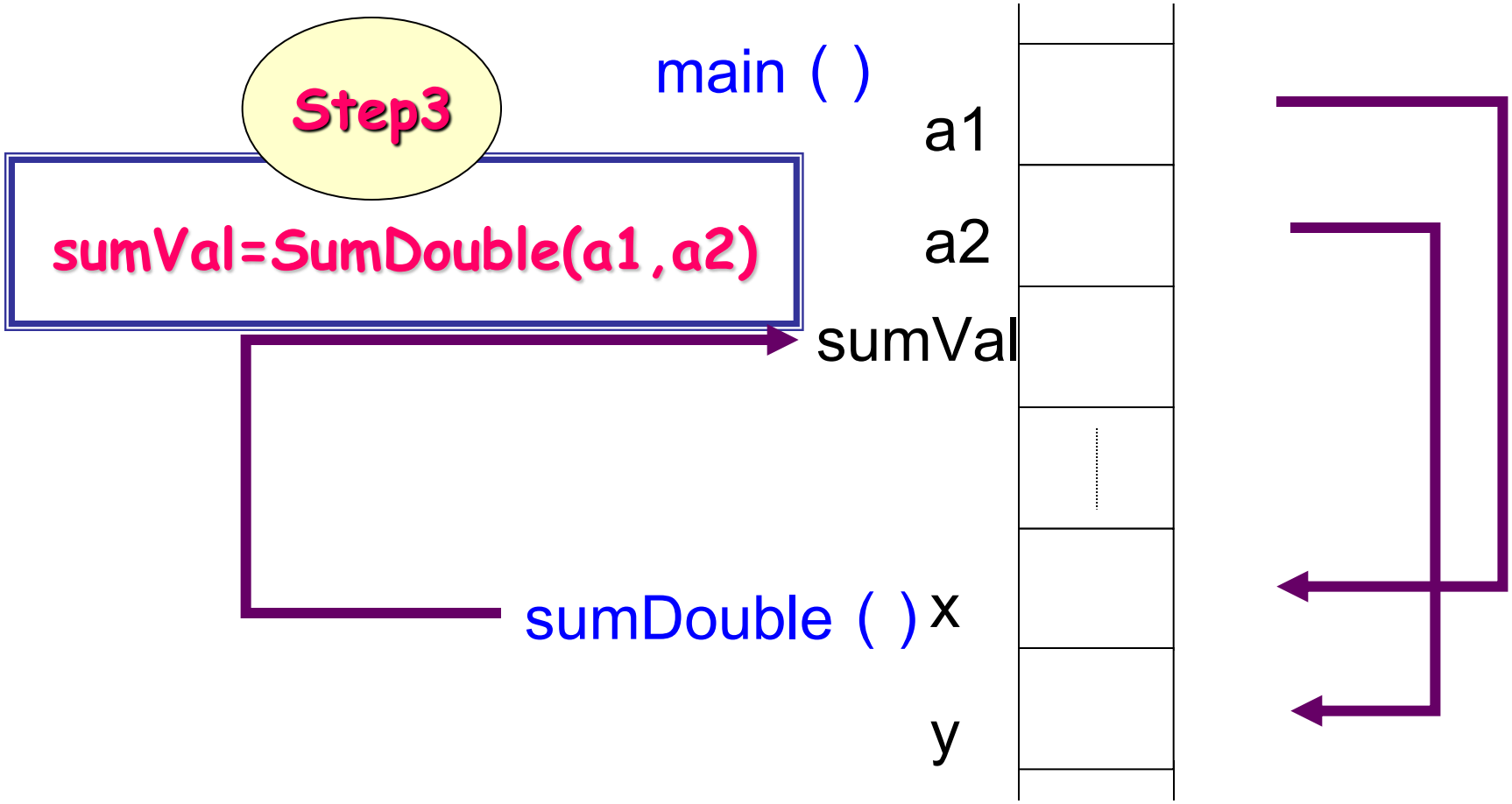
จากตัวอย่าง 4.1 และ 4.2 สามารถแสดงขอบเขตการทำงานได้ดังนี้



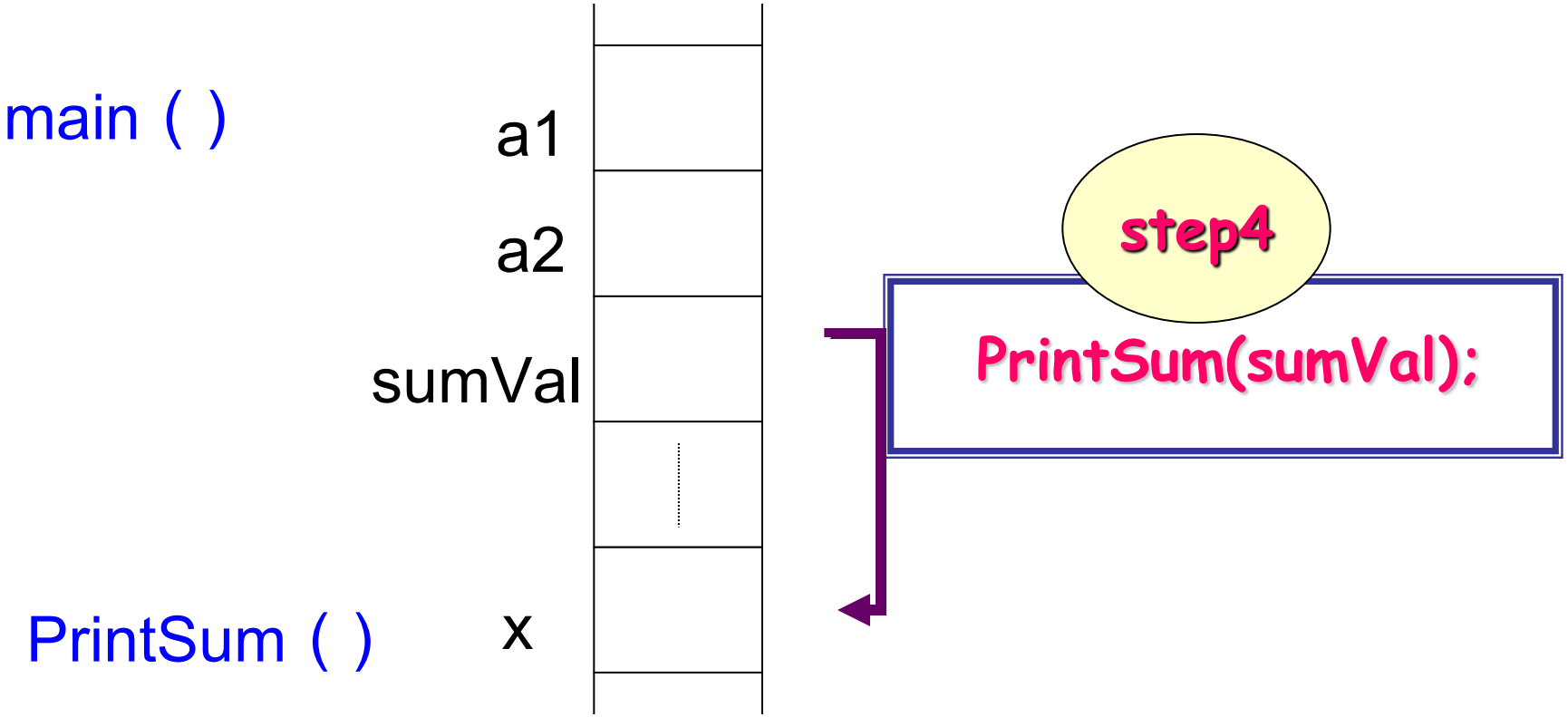
ตัวอย่าง (ต่อ)



ตัวอย่าง (ต่อ)



ตัวอย่าง (ต่อ)



จะเห็นว่าตัวแปร x ที่ประกาศในแต่ละขั้นตอนจะทำงานอยู่ภายในฟังก์ชันที่มีการประกาศค่าเท่านั้น และใช้พื้นที่ในการเก็บข้อมูลคนละส่วนกัน

ขอบเขตการทำงานของตัวแปรแต่ละตัวจะกำหนดอยู่ภายบล็อกของคำสั่งภายในเครื่องหมายปีกกา ({ }) หรือการประกาศในช่วงของการประกาศฟังก์ชัน เรียกตัวแปรเหล่านี้ว่า **ตัวแปรโลคอล** (Local Variable)

นอกจากนี้สามารถประกาศตัวแปรไว้ที่ภายนอกฟังก์ชันบริเวณส่วนเริ่มของโปรแกรมจะเรียกว่า **ตัวแปรโกลบอล** (Global Variable) ซึ่งเป็นตัวแปรที่สามารถเรียกใช้ที่ตำแหน่งใด ๆ ในโปรแกรมก็ได้ ยกเว้นในกรณีที่มีการประกาศตัวแปรที่มีชื่อเดียวกันตัวแปรโกลบอลภายในบล็อกหรือฟังก์ชัน

ตัวอย่าง 4.3

แสดงการทำงานของโปรแกรมในลักษณะที่มีตัวแปรโกลบอล แสดงขอบเขตการใช้งานของตัวแปรภายในโปรแกรม

```
#include  <stdio.h>
int   x;
void  func1 ( )
{
        x = x + 10;
        printf ( “func1 -> x : %d\n”, x );
}
```

ตัวอย่าง 4.3 (ต่อ)

```
void  func2 ( int  x )
{
        x = x + 10;
        printf ( “func2 -> x : %d\n”, x );
}
void  func3 ( ) {
        int  x=0;
        x  = x + 10;
        printf ( “func3 -> x : %d\n”, x );
}
```

ตัวอย่าง 4.3 (ต่อ)

```
void  main ( )
{
        x = 10;
        printf ( “main (start) -> x : %d\n”, x );
        func1 ( );
        printf ( “main (after func1) -> x : %d\n”, x );
        func2 ( x );
        printf ( “main (after func2) -> x : %d\n”, x);
        func3 ( );
        printf ( “main (after func3) -> x : %d\n”, x);
}
```

ตัวอย่าง 4.3 (ต่อ)

ผลการทำงาน

main (start) -> x : 10

func1 -> x : 20

main (after func1) -> x : 20

func2 -> x : 30

main (after func2) -> x : 20

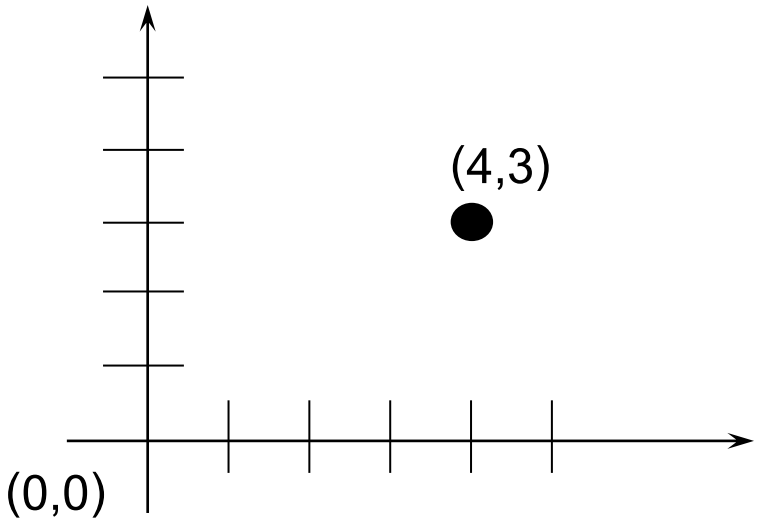
func3 -> x : 10

main (after func3) -> x : 20

ข้อมูลแบบโครงสร้างและยูเนี่ยน

(Structures and Unions)

5.1 ความรู้ทั่วไปเกี่ยวกับโครงสร้าง



หากต้องการเก็บข้อมูลจุดบนแกนโคออดิเนท จะประกอบไปข้อมูลแกน x และ y เป็นข้อมูลจำนวนเต็มประเภท int ประเภทข้อมูลที่ใช้ได้แก่ประเภทข้อมูลแบบโครงสร้าง สามารถประกาศประเภทข้อมูลที่ใช้ดังนี้

การประกาศประเภทข้อมูลแบบโครงสร้าง

```
struct    point {
                int  x;
                int  y;
          };
```

Member

หมายเหตุ *การประกาศชื่อสมาชิกภายใน struct จะใช้ชื่อใดก็ได้ อาจจะซ้ำกับชื่อตัวแปรที่อยู่ภายนอก struct แต่ชื่อที่อยู่ภายใน struct เดียวกันห้ามประกาศชื่อซ้ำกัน*

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

แบบที่ 1

```
struct    point {
                int  x;
                int  y;
        }  x, y , z;
```

หมายเหตุ *จะเห็นว่าชื่อของ struct จะประกาศหรือไม่ก็ได้ หากไม่มีการประกาศจะไม่สามารถนำ struct นั้นกลับมาใช้ได้อีก*

แบบที่ 2

```
struct    point {
                int  x;
                int  y;
          };

struct  point x,y,z
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

การกำหนดค่าเริ่มต้นให้กับตัวแปรข้อมูลแบบโครงสร้าง

```
struct  point pt = {320,200};
```

การอ้างถึงสมาชิกภายในตัวแปรข้อมูลแบบโครงสร้าง

```
struct_name.member
```

ตัวอย่าง

เมื่อต้องการอ้างถึงสมาชิกภายใน struct ว่าอยู่ตรงกับจุดใดบนแกนโคออดิเนทจะใช้

```
printf ( “%d, %d”, pt.x, pt.y );
```

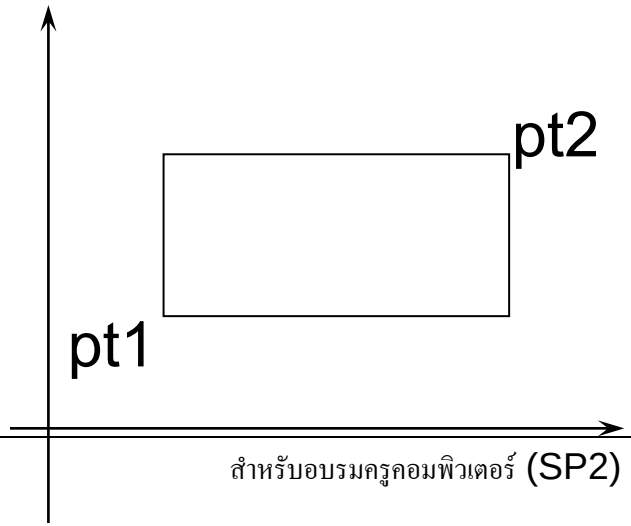
หรือหากต้องการคำนวณระยะทางจะว่าห่างจากจุดเริ่มต้น (0, 0) เท่าใดสามารถใช้

```
double  dist, sqrt (double);
```

```
dist =sqrt ((double)pt.x * pt.x +(double)pt.y * pt.y );
```

<u>หมายเหตุ</u> สมาชิกของข้อมูลประเภท struct อาจจะเป็นตัวแปรประเภทใดก็ได้ ทั้งข้อมูลพื้นฐาน และประเภทข้อมูลอื่น ๆ เช่น อาเรย์ และยังประกาศ ตัวแปรของข้อมูลประเภท struct ได้อีกด้วย

ตัวอย่าง



หากต้องการเก็บข้อมูลของสี่เหลี่ยมดังรูป สามารถทำการประกาศตัวแปรได้ดังนี้

```
struct    rect {
          struct point  pt1;
          struct point  pt2;
};

struct  rect  screen;
int co_x;
co_x = screen.pt1.x
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

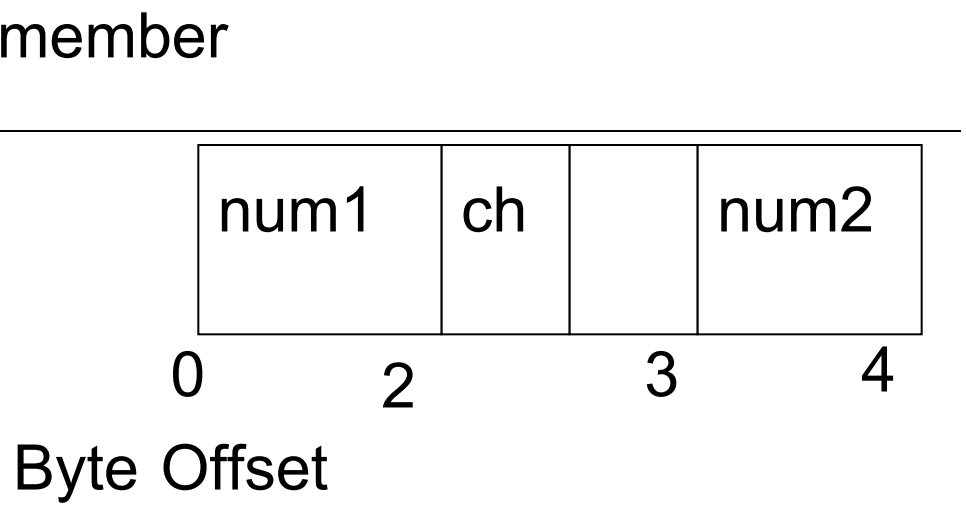
การอ้างถึงสมาชิก

5.2 การเก็บข้อมูลแบบโครงสร้าง

การเก็บข้อมูลแบบโครงสร้างภายในหน่วยความจำจะเก็บตามลำดับที่มีการประกาศสมาชิกของข้อมูลนั้น โดยทั่วไปข้อมูลแบบโครงสร้างจะประกอบขึ้นจากข้อมูลหลาย ๆ ชนิด และข้อมูลแต่ละชนิดมักจะมีการจองพื้นที่ใช้งานแต่ต่างกัน เนื่องจากการจองพื้นที่หน่วยความจำในระบบส่วนใหญ่จะจองที่แอดเดรสที่หารด้วย 2 หรือ 4 ลงตัว

ตัวอย่าง

```
struct  alignment {
        int       num1;
        char  ch;
        int       num2;
}  example;
```



จะเห็นว่า num2 จะไม่สามารถใช้พื้นที่ที่ติดกับ ch ได้ เนื่องจาก num2 เป็นข้อมูลประเภทเลขจำนวนต้องใช้พื้นที่ที่มีแอดเดรสหารด้วย 2 หรือ 4 ลงตัว ทำให้เกิดที่ว่างที่ไม่สามารถนำมาใช้ประโยชน์ได้ เพราะฉะนั้นการประกาศสมาชิกของโครงสร้างจะมีผลต่อการใช้พื้นที่ในหน่วยความจำด้วย

5.3 การใช้ข้อมูลแบบโครงสร้างกับฟังก์ชัน

การทำงานของตัวแปรที่เป็นประเภทโครงสร้างสามารถทำงานต่าง ๆ ได้เช่นเดียวกับตัวแปรอื่น ๆ ยกเว้นการเปรียบเทียบตัวแปร struct กับตัวแปร struct เนื่องจากข้อมูลของตัวแปร struct จะเก็บอยู่ในตัวแปรที่เป็นสมาชิกของ struct การเปรียบเทียบจึงต้องทำผ่านตัวแปรที่เป็นสมาชิกของ struct เท่านั้น การใช้งานตัวแปร struct กับฟังก์ชันสามารถทำได้หลายลักษณะ ทั้งการให้ฟังก์ชันคืนค่าเป็น struct การส่งอากิวเมนท์ให้ฟังก์ชันเป็นตัวแปร struct

ตัวอย่าง 5.1

ฟังก์ชันใช้ในการกำหนดค่าให้กับตัวแปร struct

```
struct  point  makepoint ( int  x, int  y )
{
    struct  point  temp;
    temp.x  =  x;
    temp.y  =  y;
    return  temp;
}
```

<u>หมายเหตุ</u> ตัวอย่างนี้แสดงฟังก์ชันที่ทำการส่งค่ากลับเป็นรูปแบบโครงสร้าง

การเรียกใช้งานฟังก์ชัน

```
struct  rect  screen;
struct  point  middle;
struct  point  makepoint ( int, int );
screen.pt1  =  makepoint ( 0, 0 );
screen.pt2  =  makepoint ( XMAX, YMAX );
middle  = makepoint ((screen.pt1.x + screen.pt2.x) / 2,
                     (screen.pt1.y + screen.pt2.y) / 2 );
```

ตัวอย่าง 5.2

ฟังก์ชันการบวก x และ y ของจุด 2 จุด และคืนค่าผลของการบวกเป็น struct

```
struct point addpoint(struct point p1, struct point p2)
{
    p1.x  +=  p2.x;
    p1.y  +=  p2.y;
    return  p1;
}
```

<u>หมายเหตุ</u> *ตัวอย่างนี้แสดงการส่งอาร์กิวเมนท์แบบ struct ให้กับฟังก์ชัน*

ตัวอย่าง 5.3

ฟังก์ชันการหาว่าจุดอยู่ในพื้นที่สี่เหลี่ยมหรือไม่

```
int  pinrect ( struct  point  p,  struct  rect  r )
{
    return  p.x >= r.pt1.x  &&  p.x < r.pt2.x  &&
            p.y >= r.pt1.y  &&  p.y < r.pt2.y;
}
```

<u>หมายเหตุ</u> *ตัวอย่างนี้เป็นการหาว่าจุดที่ระบุอยู่ในพื้นที่สี่เหลี่ยมหรือไม่ โดยส่งค่าจุดและพื้นที่สี่เหลี่ยมเป็นอากิวเมนท์ให้กับฟังก์ชัน หากจุดอยู่ในพื้นที่สี่เหลี่ยมจะคืนค่า1 แต่หากจุดอยู่นอกพื้นที่สี่เหลี่ยมจะคืนค่าเป็น 0*

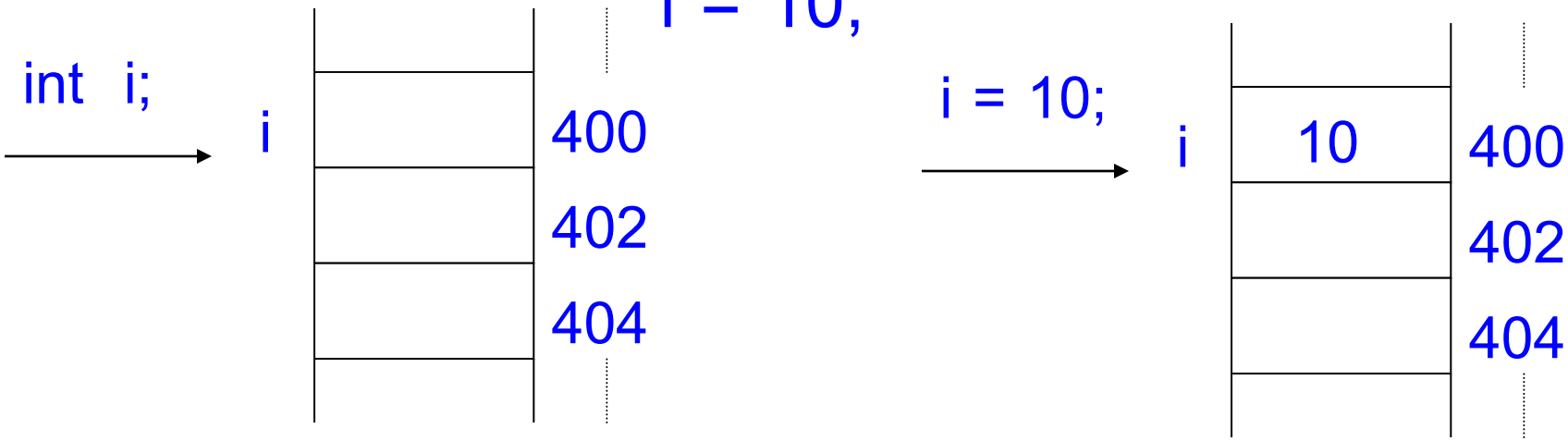
ตัวชี้และอาร์เรย์

(Pointer and Array)

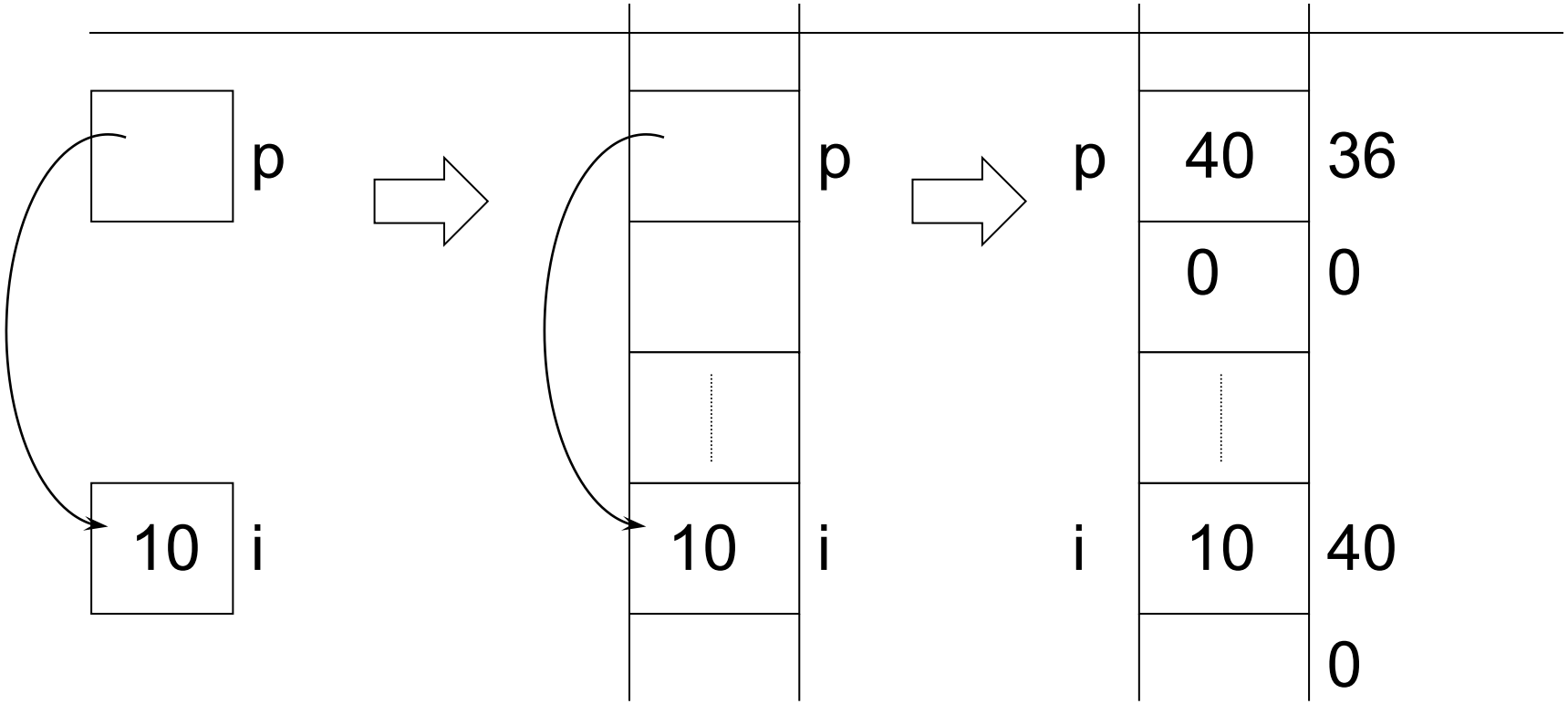
6.1 ตัวชี้กับแอดเดรส (Pointers and Address)

```
int  i;

i = 10;
```



รูปที่ 6.1 การแทนข้อมูลในหน่วยความจำของตัวแปรประเภทพื้นฐาน



รูปที่ 6.2 การแทนข้อมูลในหน่วยความจำของตัวแปรประเภทตัวชี้

6.2 การประกาศตัวแปรประเภทตัวชี้

การประกาศตัวแปรประเภทพอยน์เตอร์จะใช้ Unary Operator * ซึ่งมีชื่อเรียกว่า Indirection หรือ Dereferencing Operator โดยจะต้องประกาศประเภทของตัวแปรพอยน์เตอร์ให้สอดคล้องกับประเภทของตัวแปรที่เราต้องการ (ยกเว้นตัวแปรพอยน์เตอร์ประเภท void ที่สามารถชี้ไปยังตัวแปรประเภทใดก็ได้)

ตัวอย่าง

```
int  *ip;
```

เป็นการประกาศตัวแปร ip ให้เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท int

```
double  *dp,  atof(char  *);
```

เป็นการประกาศตัวแปร dp เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท double และประกาศฟังก์ชัน atof มีพารามิเตอร์เป็นตัวแปรพอยน์เตอร์ประเภท char

6.3 การกำหนดค่าและการอ่านค่าตัวแปรประเภทตัวชี้

การกำหนดค่าให้กับตัวแปรพอยน์เตอร์จะเป็นการกำหนดแอดเดรสของตัวแปรที่มีประเภทสอดคล้องกับประเภทของตัวแปรพอยน์เตอร์เท่านั้น โดยการใช้ Unary Operator & เป็นโอเปอเรเตอร์ที่อ้างถึงแอดเดรสของออปเจ็ค (Object) ใด ๆ

```
int  x = 1, y = 2;
int  *ip, *iq;
ip   =  &x;
y    =  *ip;
*ip  =  0;
y    = 5;
ip   =  &y;
*ip  =  3;
iq   = ip;
```

รูปที่ 6.3 การกำหนดค่าและการอ่านค่าตัวแปรตัวชี้

x	1	400
y	2	402
	.....	
ip		500
iq		502

```
int  x = 1, y = 2;
int  *ip, *iq;
```

x	1	400
y	2	402
	⋮	
ip	400	500
iq		502

```
ip = &x;
```

x	1	400
y	1	402
	⋮	
ip	400	500
iq		502

```
y = *ip;
```

x	0	400
y	1	402
	⋮	
ip	400	500
iq		502

```
*ip = 0;
```

x	0	400
y	5	402
	⋮	
ip	400	500
iq		502

y = 5;

x	0	400
y	5	402
	⋮	
ip	402	500
iq		502

```
ip = &y;
```

x	0	400
y	3	402
	⋮	
ip	402	500
iq		502

```
*ip = 3;
```

x	0	400
y	3	402
	⋮	
ip	402	500
iq	402	502

```
iq =  ip;
```

6.4 ตัวชี้และอาร์กิวเมนท์ของฟังก์ชัน

(Pointer and Function Arguments)

เนื่องจากภาษาซีมีการส่งอากิวเมนต์ให้กับฟังก์ชันแบบ By Value และฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่า หากต้องการให้ฟังก์ชันมีการเปลี่ยนแปลงค่าและคืนค่ากลับมายังฟังก์ชันที่เรียกใช้มากกว่าหนึ่งค่าจะต้องนำพอยน์เตอร์เข้ามาช่วย

ตัวอย่างเช่น หากต้องการเขียนฟังก์ชันเพื่อสลับค่าของตัวแปร 2 ตัว ผลลัพธ์ที่ต้องการได้จากฟังก์ชันนี้จะมี 2 ค่าของตัวแปรที่ทำการสลับค่า หากอาร์กิวเมนต์เป็นตัวแปรธรรมดาจะไม่สามารถแก้ปัญหานี้ได้ จึงต้องใช้พอยน์เตอร์เข้ามาช่วย โดยการส่งค่าแอดเดรสของตัวแปรทั้ง 2 ให้กับฟังก์ชันที่จะสลับค่าของตัวแปรทั้ง 2 ผ่านทางตัวแปรพอยน์เตอร์ที่เป็นอาร์กิวเมนต์ของฟังก์ชัน

ตัวอย่าง 6.1

โปรแกรมตัวอย่างการสลับค่าตัวแปร 2 ตัว โดยผ่านฟังก์ชัน จะแสดงการส่งอาร์กิวเมนต์ในเป็นพอยน์เตอร์

```
#include <stdio.h>
void swap (int *, int *);
```

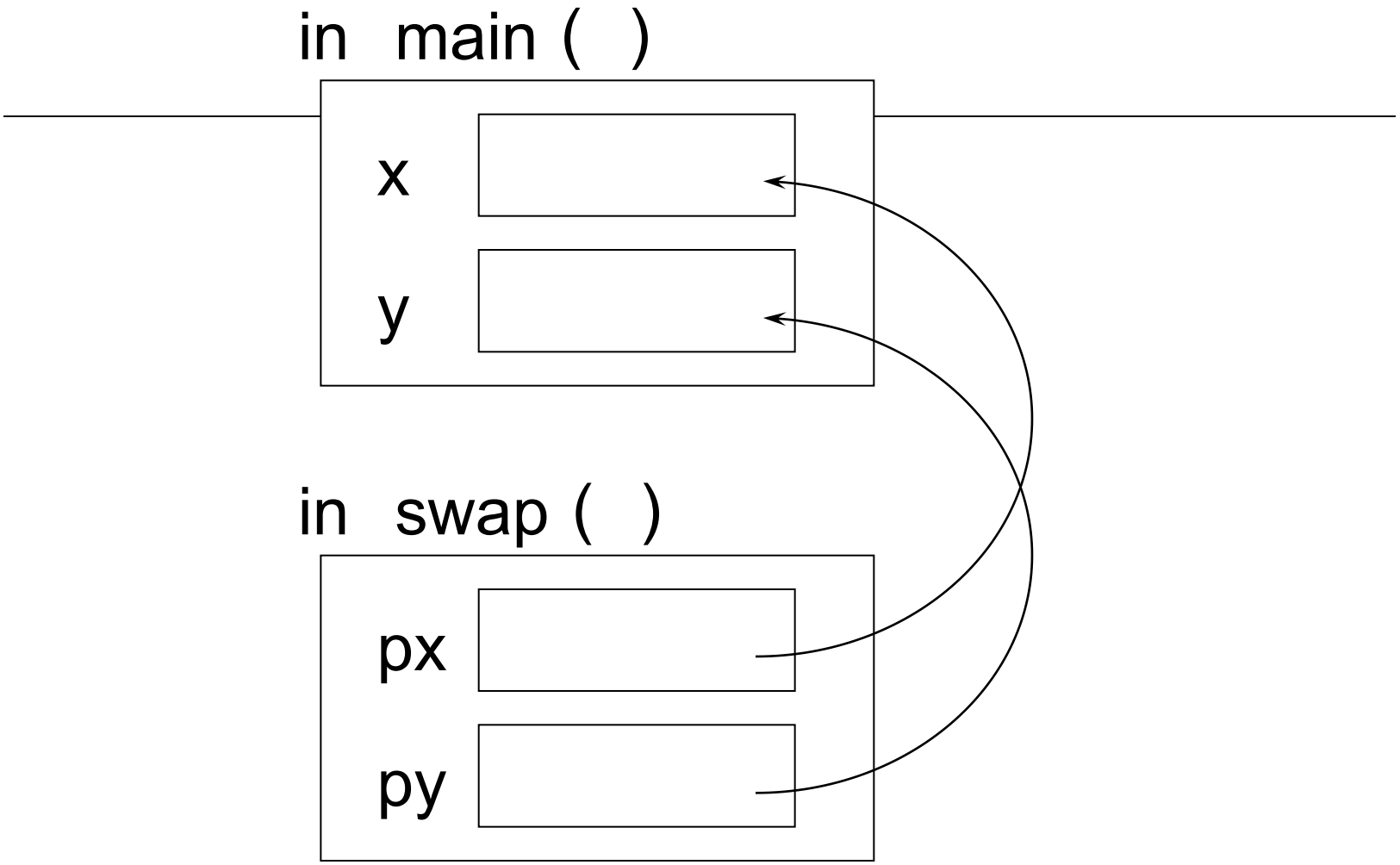
ตัวอย่าง 6.1 (ต่อ)

```
void  main ( )
{
    int  x = 5,  y = 10;
    printf("Before swap : x = %d", x, ", y = %d\n", y);
    swap ( &x,  &y);
    printf("After swap : x = %d", x, ", y = %d\n", y);
}
```

ตัวอย่าง 6.1 (ต่อ)

```
void  swap (int  *px,  int  *py)
{
    int  temp;
    temp  =  *px;
    *px     = *py;
    *py     =  temp;
}
```

อาร์กิวเมนท์ที่เป็นประเภทพอยน์เตอร์จะช่วยให้ฟังก์ชันสามารถเปลี่ยนค่าให้กับตัวแปรที่ส่งเข้ามาได้ เนื่องจากอาร์กิวเมนท์นั้นจะเก็บแอดเดรสของตัวแปรที่ส่งเข้ามา เมื่อมีการเปลี่ยนแปลงค่าของอาร์กิวเมนท์ผ่าน Dereferencing Operator (*) ค่าของตัวแปรที่ส่งเข้ามาจะถูกเปลี่ยนค่าพร้อมกันในทันที



รูปที่ 6.4 แสดงความสัมพันธ์ของการส่งอาร์กิวเมนท์แบบพอยน์เตอร์กับฟังก์ชัน

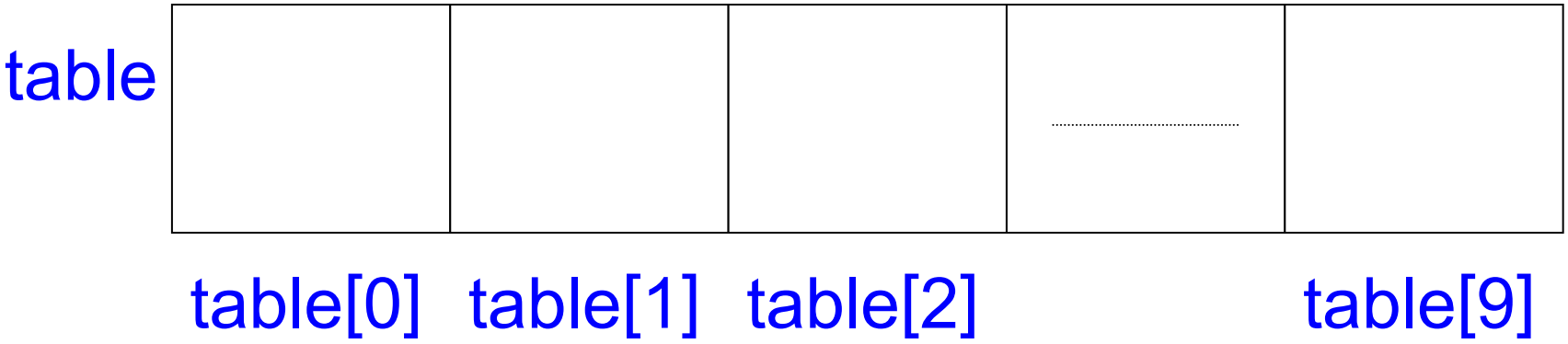
6.5 ตัวชี้กับอาร์เรย์ (Pointer and Arrays)

อาร์เรย์เป็นประเภทข้อมูลที่เก็บชุดของข้อมูลประเภทเดียวกัน มักใช้กับการทำงานที่ต้องทำงานกับตัวแปรชนิดเดียวกันหลายตัวที่มีการทำงานเหมือนกัน เช่น คะแนนของนักศึกษาภายในห้อง 20 คน เป็นต้น อาร์เรย์ในภาษาซีจะนำหลักการของพอยน์เตอร์เข้ามาใช้ การทำงานใด ๆ ของอาร์เรย์สามารถใช้พอยน์เตอร์เข้ามาแทนที่

การประกาศอาร์เรย์

int table[10];

เป็นการกำหนดอาร์เรย์ชื่อ table เป็นอาร์เรย์ประเภท int ที่มีสมาชิกทั้งหมด 10 ตัว ตั้งแต่ table[0], table[1], table[2], ... , table[9] สมาชิกภายในอาร์เรย์จะเริ่มที่ 0 เสมอ และสมาชิกตัวสุดท้ายจะอยู่ที่ตำแหน่งของขนาดที่ประกาศไว้ลบด้วย 1



รูปที่ 6.5 แสดงภาพจำลองของอาร์เรย์ขนาดสมาชิก 10 ตัว

การอ้างถึงสมาชิกในอาร์เรย์

จะใช้ระบบดัชนีโดยผ่านเครื่องหมาย [ ] เช่น อ้างถึงสมาชิกตัวที่ 3 ของอาร์เรย์ด้วย table[2] เป็นต้น การใช้งานสมาชิกของอาร์เรย์สามารถใช้งานได้เหมือนตัวแปรพื้นฐานทั่วไป

ตัวอย่าง

```
sumThird  =  table[0] + table[1] + table[2];
table[0]    =  5;
if ( a[0] > a[9] )
        printf (“First is greater than last\n” );
```

เราสามารถอ้างถึงสมาชิกทุกตัวภายในอาร์เรย์อย่างอิสระ ภายในขอบเขตของขนาดที่ได้ประกาศอาร์เรย์ไว้ แต่การใช้อาร์เรย์มักจะเป็นการเข้าถึงสมาชิกในลักษณะทั่วไปโดยใช้ตัวแปรประเภท int มาช่วย

ตัวอย่าง

สมมติให้ i, j, k เป็นตัวแปรประเภท int

```
for (int k = 0; k < 9; k++)
        printf (“Value at %d  = %d\n”, k+1, table[k]);
table[i + j] = 0;
table[7 – table[j]] = j;
```

สิ่งที่ต้องระวัง

ในภาษาซีจะไม่มีการกำหนดให้ตรวจสอบขอบเขตของอาร์เรย์ โปรแกรมเมอร์จะต้องพยายามเขียนโปรแกรมที่เกี่ยวข้องกับสมาชิกของอาร์เรย์ภายในขอบเขตที่ประกาศอาร์เรย์ไว้ หากมีการอ้างอิงถึงสมาชิกอาร์เรย์นอกขอบเขตที่ได้ระบุไว้ เช่น table[12] สิ่งที่ได้คือการไปอ่านข้อมูลในพื้นที่ของหน่วยความจำที่อาจจะเก็บค่าของตัวแปรตัวอื่น หรือค่าอื่นใดที่ไม่อาจคาดเดาได้

ตัวอย่าง 6.2

ให้อ่านค่าของจำนวนเต็ม 5 จำนวนจากคีย์บอร์ด และแสดงผลในลำดับที่กลับกัน

```
# include  <stdio.h>
#define  SIZE  5
main ( ) {
        int  k;
        int  table[SIZE];
        for  (k = 0;  k < SIZE;  k++)
                scanf (“%d”, &table[k]);
        for (k = SIZE-1; k >= 0;  k--)
                printf (“%d\n”, table[k]);
}
```

สมาชิกของอาร์เรย์อาจเป็นประเภทข้อมูลพื้นฐานใด ๆ ก็ได้ หรืออาจเป็นข้อมูลประเภท Enumeration เช่น

```
#define  TSIZE        10
#define  NAMESIZE     20
#define  ADDRSIZE     30
enum  month { JAN, FEB, MAR, APR, MAY,
              JUN, JUL, AUG, SEP, OCT,
              NOV, DEC }
```

```
typedef enum month Month;
int age[TSIZE];
float size[TSIZE+1];
Month date[8];
char name[NAMESIZE], address[ADDRSIZE];
```

6.6 การใช้ตัวชี้กับอาร์เรย์

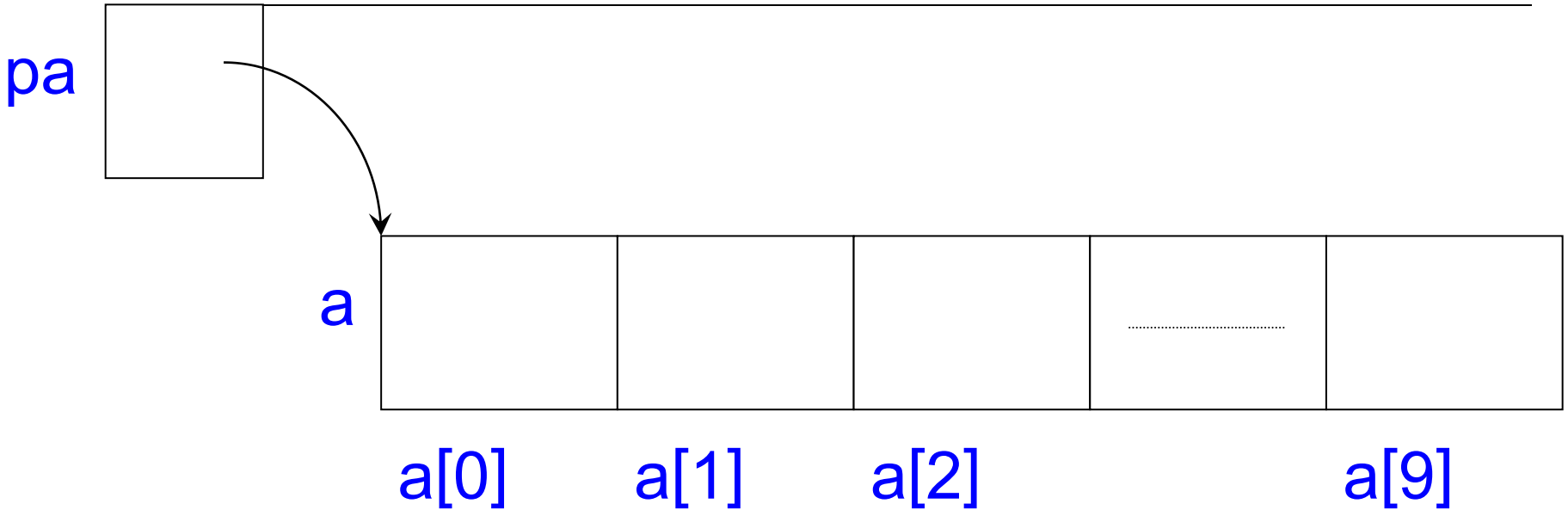
การทำงานใด ๆ ของอาร์เรย์สามารถใช้พอยน์เตอร์เข้ามาช่วย ซึ่งจะทำให้มีความเร็วในการทำงานสูงขึ้น สมมติว่ามีอาร์เรย์ a และพอยน์เตอร์ pa ดังนี้

```
int   a[10];
int   *pa;
```

กำหนดให้พอยน์เตอร์ pa ชี้ไปยังอาร์เรย์ a ด้วยคำสั่ง

```
pa = &a[0];  /*  หรือใช้คำสั่ง  pa = a;  */
```

pa จะเก็บค่าแอดเดรสเริ่มต้นของอาร์เรย์ a



รูปที่ 6.6 แสดงตัวชี้ชี้ไปยังแอดเดรสเริ่มต้นของอาร์เรย์

การนำไปใช้งานจะสามารถอ่านค่าอาร์เรย์

ผ่านพอยน์เตอร์ได้ดังนี้

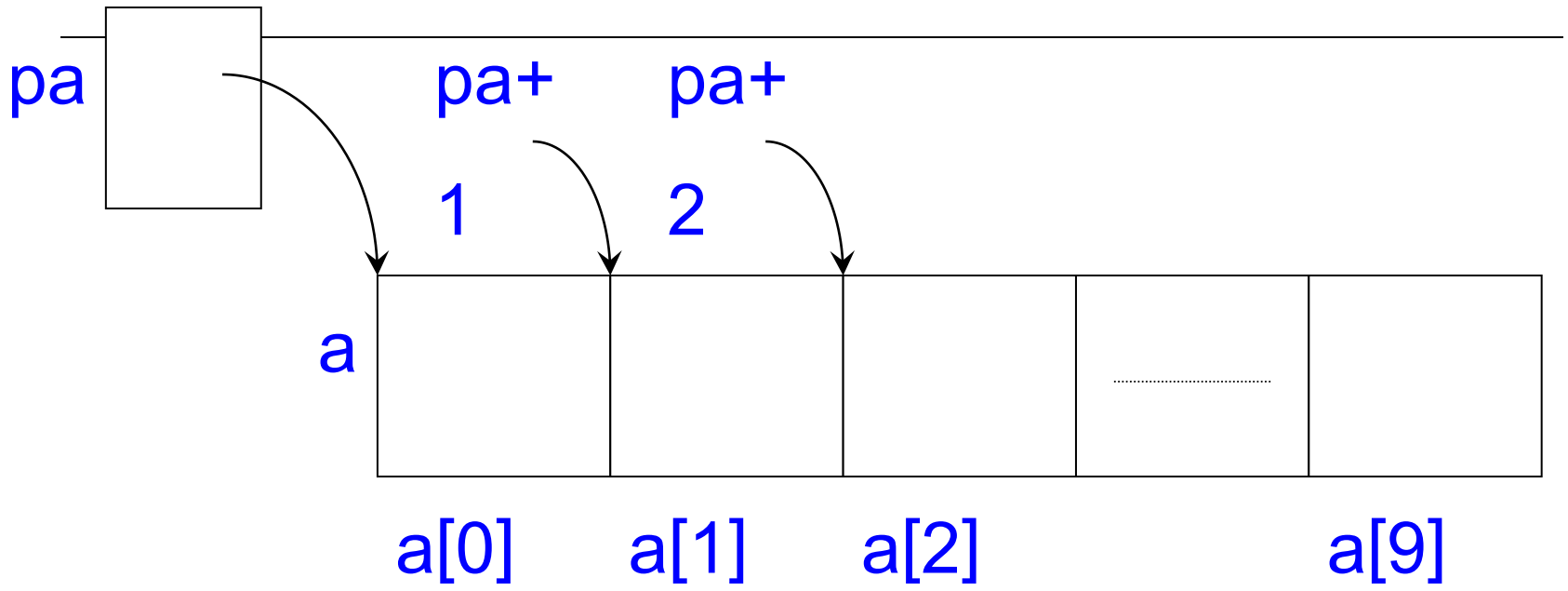
```
x  =  *pa;
```

จะเป็นการกำหนดค่าให้ x มีค่าเท่ากับ a[0] การเลื่อนไปอ่านค่าสมาชิกตำแหน่งต่าง ๆ ของอาร์เรย์ผ่านทางพอยน์เตอร์สามารถทำได้โดยการเพิ่มค่าพอยน์เตอร์ขึ้น 1 เพื่อเลื่อนไปยังตำแหน่งถัดไป หรือเพิ่มค่าขึ้น N เพื่อเลื่อนไป N ตำแหน่ง หรืออาจจะลดค่าเพื่อเลื่อนตำแหน่งลง

กรณีที่ pa ชี้อยู่ที่ a[0] คำสั่ง

pa+1;

จะเป็นการอ้างถึงแอดเดรสของ a[1] หากเป็น pa+i เป็นการอ้างถึงแอดเดรส a[i] หากต้องการอ้างถึงข้อมูลภายในของสมาชิกของอาร์เรย์ตำแหน่งที่ a[i] จะใช้ *(pa+i)



รูปที่ 6.7 แสดงการอ้างถึงตำแหน่งในอาร์เรย์ผ่านตัวชี้

การสั่งให้บวก 1 หรือบวก i หรือ ลบ i เป็นเหมือนการเลื่อนไปยังสมาชิกของอาร์เรย์ตำแหน่งที่ต้องการ เนื่องจากประเภทของข้อมูลแต่ละประเภทของอาร์เรย์ เช่น int, float, double และอื่น ๆ มีขนาดของข้อมูลที่ต่างกัน ทำให้ขนาดของสมาชิกภายในอาร์เรย์แต่ละประเภทมีขนาดแตกต่างกันด้วย การสั่งให้บวกหรือลบด้วยจำนวนที่ต้องการนั้นจะมีกลไกที่ทำหน้าที่คำนวณตำแหน่งที่ต้องการให้สอดคล้อง กับข้อมูลแต่ละประเภทโดยอัตโนมัติ

นอกจากนี้ยังสามารถใช้พอยน์เตอร์แทนอาร์เรย์ การอ้างโดยใช้ a[i] สามารถใช้ *(a+i) เนื่องจากทุกครั้งที่อ้างถึง a[i] ภาษาซีจะทำหน้าที่แปลงเป็น *(a+i) เพราะฉะนั้นการเขียนในรูปแบบใดก็ให้ผลลัพธ์ในการทำงานเช่นเดียวกัน และการอ้างถึงแอดเดรส เช่น &a[i] จะมีผลเท่ากับการใช้ a+i

ในลักษณะเดียวกันการใช้งานพอยน์เตอร์ก็สามารถใช้คำสั่งในลักษณะอาร์เรย์ก็ได้ เช่น การอ้างถึง *(pa+i) สามารถเขียนด้วย pa[i] ก็ได้ผลเช่นเดียวกัน

สิ่งที่แตกต่างกันของอาร์เรย์และพอยน์เตอร์ คือ พอยน์เตอร์เป็นตัวแปร แต่อาร์เรย์ไม่ใช่ตัวแปร สมมติให้ a เป็นอาร์เรย์และ pa เป็นพอยน์เตอร์ การอ้างถึง pa = a หรือ pa++ จะสามารถคอมไพล์ได้ แต่จะไม่สามารถใช้คำสั่ง a = pa หรือ a++ ได้

เมื่อมีการส่งชื่อของอาร์เรย์ให้แก่ฟังก์ชัน จะเป็นการส่งตำแหน่งแอดเดรสของสมาชิกตัวแรกของอาร์เรย์ให้แก่ฟังก์ชัน ดังนั้นพารามิเตอร์ในฟังก์ชันนั้นจะเป็นตัวแปรประเภทพอยน์เตอร์

ตัวอย่าง 6.3

ฟังก์ชันที่รับพารามิเตอร์เป็นพอยน์เตอร์โดยอาร์กิวเมนท์ที่ส่งมาเป็นอาร์เรย์

```
int  strlen (char *s)
{
    int  n;
    for ( n = 0; *s != '\0'; s++ )
        n++;
    return  n;
}
```

จะเห็นว่า s เป็นพอยน์เตอร์ ในฟังก์ชันจะมีการตรวจสอบข้อมูลว่ามีค่าเท่ากับ '\0' หรือไม่ และมีการเลื่อนตำแหน่งทีละ 1 ค่า (นับว่าข้อมูลมีความยาวเพิ่มขึ้นทีละ1) โดยใช้ s++

การเรียกใช้ฟังก์ชัน strlen สามารถทำได้หลายลักษณะ

```
strlen ("hello world");        /* string constant */
strlen (array);                /* char  array[10]  */
strlen (ptr);                  /*  char  *ptr;        */
```

นอกจากนี้ยังอาจจะประกาศพารามิเตอร์ภายในฟังก์ชัน strlen ได้ใน 2 ลักษณะ คือ char *s แบบในตัวอย่าง หรืออาจจะใช้ char s[ ] ก็ได้ โดยทั่วไปจะใช้ในลักษณะแรก เพราะช่วยในรู้ได้ทันทีว่า s เป็นตัวแปรพอยน์เตอร์ และยังสามารถส่งส่วนใดส่วนของอาร์เรย์ให้แก่ฟังก์ชันก็ได้ โดยไม่จำเป็นต้องส่งสมาชิกตัวแรกก็ได้เช่นกัน

ตัวอย่าง

f (&a[2])

หรือ f (a+2)

เป็นการส่งแอดเดรสของสมาชิก a[2] ให้กับฟังก์ชัน f การประกาศฟังก์ชัน f สามารถทำได้โดยการประกาศ

f (int arr[ ]) { ..... }

หรือ f (int *arr) { ..... }

6.7 การคำนวณกับแอดเดรส

ให้ p เป็นพอยน์เตอร์ชี้ไปยังอาร์เรย์ใด ๆ คำสั่ง p++ เป็นการเลื่อน p ไปยังสมาชิกถัดไป และคำสั่ง p += i เป็นการเลื่อนพอยน์เตอร์ไป i ตำแหน่งจากตำแหน่งปัจจุบัน นอกจากนี้ยังสามารถใช้เครื่องหมายความสัมพันธ์ (Relational Operator) เช่น ==, !=, <, >= และอื่น ๆ ทำงานร่วมกับพอยน์เตอร์ได้ สมมติให้ p และ q ชี้ไปยังสมาชิกของอาร์เรย์เดียวกัน

$$p < q$$

จะเป็นจริงเมื่อ p ชี้ไปที่สมาชิกที่อยู่ก่อนหน้าสมาชิกที่ q ชี้อยู่ การเปรียบเทียบในลักษณะจะใช้ได้ต่อเมื่อ p และ q ชี้ไปที่อาร์เรย์เดียวกันเท่านั้น

นอกจากนี้ยังสามารถใช้การลบหรือการบวกกับพอยน์เตอร์ได้เช่นเดียวกัน แต่สิ่งที่ควรระวังคือการทำเช่นนั้นจะต้องอยู่ในขอบเขตขนาดของอาร์เรย์เท่านั้น

ตัวอย่าง 6.3

ฟังก์ชัน strlen( ) ปรับปรุงให้กระชับขึ้น

```
int  strlen (char *s)
{
    char  *p = s;
    while  (*p != '\0')
        p++;
    return  p-s;
}
```

เนื่องจาก s ชี้อยู่ที่ตำแหน่งเริ่มต้น โดยมี p ชี้ไปที่ s เช่นเดียวกัน แต่จะมีการเลื่อน p ไปทีละหนึ่งตำแหน่ง จนกว่าค่าที่ตำแหน่งที่ p ชี้อยู่จะเท่ากับ '\0' เมื่อนำ p ค่าสุดท้ายมาลบกับ s ที่ตำแหน่งเริ่มต้นก็จะได้ความยาวของข้อมูลที่ส่งเข้ามา

6.8 ตัวชี้ตัวอักษรและฟังก์ชัน (Character Pointer and Function)

การทำงานกับข้อความหรือที่เรียกว่า สตริง (String) เป็นการใช้ข้อมูลตัวอักษรหลาย ๆ ตัว หรืออาร์เรย์ของข้อมูลประเภท char หรืออาจจะใช้พอยน์เตอร์ชี้ไปยังข้อมูลประเภท char การทำงานกับค่าคงที่สตริง (String Constant) สามารถเขียนภายในเครื่อง “ ”

ตัวอย่าง

"I am a string"

เมื่อมีการใช้ค่าคงที่สตริงจะมีการพื้นที่ในหน่วยความจำเท่ากับความยาวของค่าคงที่สตริงบวกด้วย 1 เนื่องจากลักษณะการเก็บข้อมูลประเภทข้อความในหน่วยความจำจะมีการปะตัวอักษร null หรือ '\0' ต่อท้ายเสมอเพื่อให้รู้ว่าเป็นจุดสิ้นสุดของข้อมูล การจองพื้นที่ดังกล่าวจะเหมือนการจองพื้นที่ของข้อมูลประเภทอาร์เรย์ เป็นอาร์เรย์ของ char

I		a	m		a		s	t	r	i	n	g	\0

รูปที่ 6.8 แสดงแบบจำลองการเก็บข้อมูลประเภทสตริงในหน่วยความจำ

ค่าคงที่สตริงที่พบเห็นได้เสมอได้แก่ข้อความที่ใช้ในฟังก์ชัน printf ( ) เช่น

```
printf ( “Hello, world\n” );
```

ฟังก์ชัน printf ( ) จะรับพารามิเตอร์เป็นพอยน์เตอร์ชี้ไปยังแอดเดรสของข้อมูลที่ตำแหน่งเริ่มต้นของอาร์เรย์ และนำข้อความนั้นแสดงออกทางอุปกรณ์แสดงข้อมูลมาตรฐาน

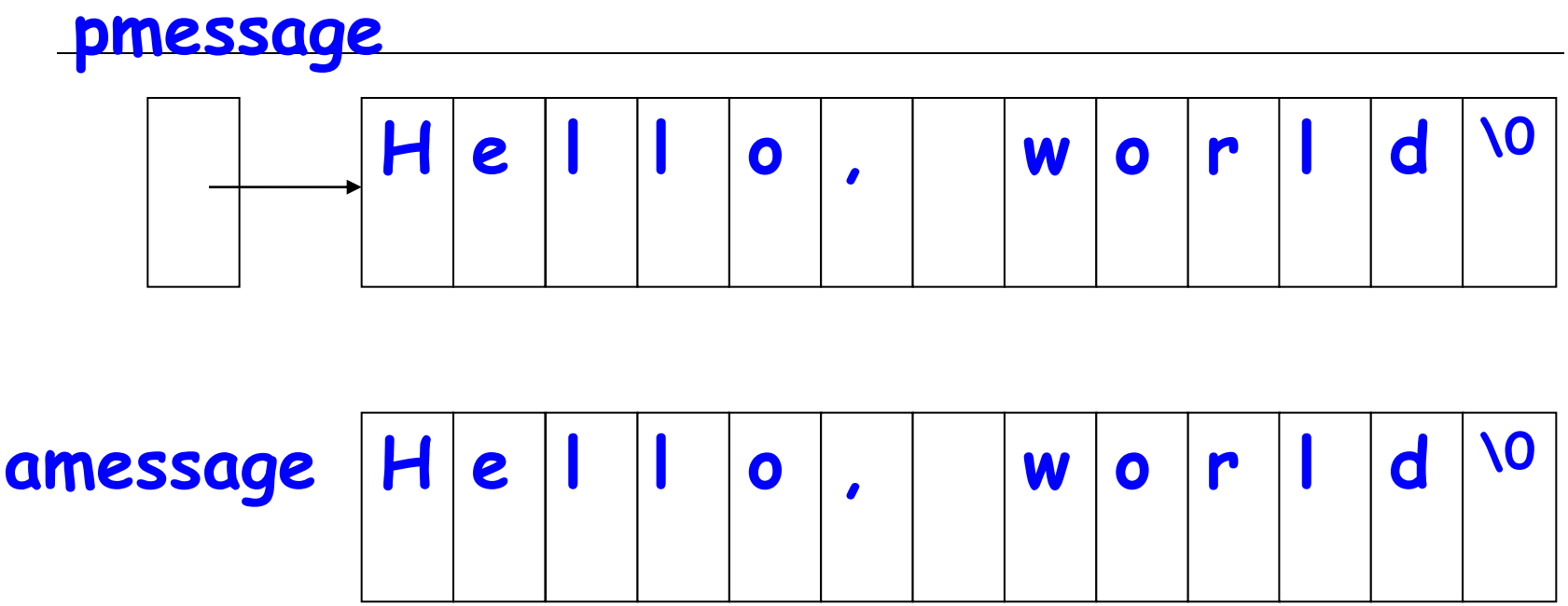
ในการเขียนโปรแกรมจะสามารถใช้พอยน์เตอร์ชี้ไปค่าคงที่สตริงใด ๆ ก็ได้ เช่น

```
char  *pmessage = “Hello, world”;
```

pmessage จะเป็นพอยน์เตอร์ประเภท char ชี้ไปที่อาร์เรย์ของตัวอักษร จะแตกต่างจากการใช้อาร์เรย์ทั่วไปเช่น

```
char  amessage[ ] = “Hello, world”;
```

ลักษณะของอาร์เรย์เช่น amessage จะมีการจองพื้นที่ใช้กับอาร์เรย์ขนาด 13 ตัวอักษร รวมทั้ง null ส่วนลักษณะของพอยน์เตอร์ที่ชี้ไปยังค่าคงที่สตริง จะมีการจองพื้นที่ให้กับค่าคงที่สตริงขนาด 13 ตัวอักษรเช่นเดียวกัน แต่จะมีการจองพื้นที่ให้กับพอยน์เตอร์และทำการชี้พอยน์เตอร์นั้นไปยังพื้นที่ของค่าคงที่สตริงที่จองเอาไว้



รูปที่ 6.9 การจองพื้นที่ให้กับอาร์เรย์และตัวชี้ชี้ไปยังค่าคงที่สตริง

ตัวอย่าง 6.5

ฟังก์ชัน strcpy ( ) ทำหน้าที่สำเนาข้อความจากตัวแปรหนึ่งไปยังอีกตัวแปรหนึ่งเขียนในลักษณะอาร์เรย์

```
void    strcpy ( char  *s,  char  *t )
{
        int  i=0;
        while ( ( s[i] = t[i] )  !=  '\0' )
                i++;
}
```

ตัวอย่าง 6.6

ฟังก์ชัน strcpy ( ) เขียนในลักษณะพอยน์เตอร์

```
void strcpy ( char  *s,  char  *t )
{
    while ( ( *s = *t )  !=  '\0' )  {
        s++;
        t++;
    }
}
```

ตัวอย่าง 6.7

ฟังก์ชัน strcpy ( ) เขียนในลักษณะพอยน์เตอร์แบบสั้น

```
void  strcpy ( char  *s,  char  *t )
{
    while ( ( *s++ = *t++ ) != '\0' )  ;
}
```

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

กรณีการส่งอากิวเมนท์เป็นตัวแปร struct จะไม่เหมาะกับ struct ที่มีขนาดใหญ่ เนื่องจากทุกครั้งที่ส่งตัวแปร struct จะเป็นการสำเนาตัวแปรตัวใหม่ขึ้นมาในฟังก์ชัน ซึ่งจะทำให้ช้าและเปลืองพื้นที่หน่วยความจำ เราจะใช้พอยน์เตอร์เข้ามาช่วยแก้ปัญหานี้

โดยส่งแอดเดรสของตัวแปร struct มายังฟังก์ชันซึ่งรับอากิวเมนท์เป็นพอยน์เตอร์ อากิวเมนท์จะชี้ไปยังแอดเดรสเริ่มต้นของตัวแปร struct จะช่วยให้การทำงานเร็วขึ้นและเปลืองหน่วยความจำน้อยลง แต่สิ่งที่ต้องระวังคือหากมีการเปลี่ยนแปลงค่าที่อากิวเมนท์พอยน์เตอร์ชี้อยู่ ค่าในตัวแปร struct ที่ส่งมายังฟังก์ชันจะเปลี่ยนตามโดยอัตโนมัติ

ตัวอย่าง

```
struct  point  origin, *pp;
pp = &original;
printf ( “origin is (%d, %d)\n”, (*pp).x, (*pp).y );
```

จะได้ตัวแปร pp ชี้ไปยังข้อมูลแบบโครงสร้างชื่อ struct point การเขียน *pp จะเป็นการอ้างถึงโครงสร้าง

การอ้างถึงสมาชิกสามารถทำได้โดยอ้าง

(*pp).x หรือ (*pp).y

<u>หมายเหตุ</u> สิ่งที่ต้องระวังคือ (*pp).x จะไม่เหมือนกับ *pp.x เนื่องจากเครื่องหมาย . จะมีลำดับความสำคัญสูงกว่า * ทำจะการแปลความหมาย *pp.x จะเหมือนกับการอ้าง *(pp.x) ซึ่งจะทำให้เกิดความผิดพลาดขึ้น

การอ้างถึงสมาชิกอาจเขียนอีกลักษณะหนึ่งโดยใช้เครื่องหมาย -> สมมติ p เป็นพอยน์เตอร์ รูปแบบการใช้เป็นดังนี้

```
p->member-of-structure
```

จะสามารถแปลงประโยคการใช้พอยน์เตอร์อ้างสมาชิกของ struct จากตัวอย่างข้างบนได้ว่า

```
printf ( “origin is (%d, %d)\n”, pp->x, pp->y);
```

ตัวอย่าง

หากมีพอยน์เตอร์ชี้ไปยัง struct rect ดังนี้

```
struct  rect  r,  *rp = r;
```

การอ้างถึงสมาชิกต่อไปนี้จะมีผลเท่ากับการอ้างถึงสมาชิกตัวเดียวกัน

```
r.pt1.x
rp->pt1.x
(r.pt1).x
(rp->pt1).x
```

6.9 ตัวชี้ (pointer) ชี้ไปยังโครงสร้าง (pointer to structures)

พอยน์เตอร์เป็นตัวแปรที่เก็บแอดเดรสของตัวแปรอื่น สามารถใช้ชี้ไปยังข้อมูลประเภทใด ๆ การใช้พอยน์เตอร์ชี้ไปยังโครงสร้างสามารถทำได้ดังนี้

แบบที่ 1

```
typedef struct {
        int day;
        int month;
        int year;
    } Date;

Date today;

Date *ptrdate;
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศตัวแปรข้อมูลแบบโครงสร้าง

การประกาศตัวแปร pointer ชี้ไปยัง โครงสร้าง

แบบที่ 2

```
struct  date {
        int  day;
        int  month;
        int  year;
    }  *ptrdate;
```

แบบที่ 3

```
typedef struct {
        int day;
        int month;
        int year;
    } Date;

typedef Date *PtrDate;

PtrDate ptrdate;
```

การประกาศแบบข้อมูลโครงสร้าง

การประกาศประเภทตัวแปร pointer ชี้ไปยังโครงสร้าง

การประกาศตัวแปร pointer ชี้ไปยัง โครงสร้าง

การประกาศตัวแปร ptrdate ทั้ง 3 ลักษณะจะสามารถใช้งานได้เหมือนกันทั้งหมด

หากต้องการให้ ptrdate ชี้ไปยังตัวแปรโครงสร้างสามารถทำได้ดังนี้

```
ptrdate = &today;
```

การอ้างถึงสมาชิกของโครงสร้างผ่านตัวแปรพอยน์เตอร์

```
ptrdate->day  = 7;
if  ( ptrdate->day  ==  31  &&  ptrdate->month  ==  12 )  .....
                    scanf ( “%d”, &ptrdate->year );
```

การอ้างถึงสมาชิกโครงสร้างโดยใช้เครื่องหมาย ->

```
(*ptrdate).day  = 7;
if ( (*ptrdate).day  ==  31  &&  (*ptrdate).month  == 12 ) .....
                    scanf ( “%d”, &((*ptrdate).year) );
```

ตัวอย่าง 6.8

โปรแกรมตัวอย่างการใช้ตัวชี้ (pointer) ชี้ไปยังโครงสร้าง

```
#include  <stdio.h>
struct  date  {        /*date template */
      int    day;
      int    month;
      int    year;
};
typedef  struct  date    Date;
typedef  Date    *PtrDate;
```

ตัวอย่าง 6.8 (ต่อ)

```
main ( ) {
    Date      today;
    PtrDate   ptrdate;
    ptrdate = &today;
    ptrdate->day      =  27;
    ptrdate->month  =  9;
    ptrdate->year     = 1985;
    printf ( “Today\’s date is %2d/%2d/%4d\n”,
       ptrdate->day, ptrdate->month, ptrdate->year );
}
```

นอกจากนี้ยังสามารถทำการกำหนดค่าเริ่มต้นให้กับตัวแปรแบบโครงสร้าง เช่น

```
Date  xmas = { 25, 12, 1986 };
```

และหากมีการกำหนดค่าเริ่มต้นให้กับสมาชิกของโครงสร้างไม่ครบทุกตัว หากตัวแปรนั้นเป็น external หรือ static ค่าของสมาชิกที่ขาดไปจะถูกกำหนดให้เป็น 0 แต่หากเป็นประเภท automatic จะไม่สามารถคาดได้ว่าค่าของสมาชิกที่ไปจะเป็นค่าใด

6.10 อาร์เรย์ของโครงสร้าง

การใช้งานโครงสร้างนอกจากใช้ในลักษณะของตัวแปรแล้ว ยังสมารถใช้งานในลักษณะของอาเรย์ได้อีกด้วย เช่น การเก็บข้อมูลประวัติของพนักงาน จะมีโครงสร้างที่ใช้เก็บข้อมูลของพนักงานแต่ละคน หากใช้ในลักษณะของตัวแปรปกติจะสามารถเก็บข้อมูลของพนักงานได้เพียง 1 คน ซึ่งพนักงานทั้งบริษัทอาจจะมีหลายสิบหรือหลายร้อยคน การเก็บข้อมูลในลักษณะนี้จะใช้อาเรย์เข้ามาช่วย เช่น

```
Person  staff[STAFFSIZE];
```

การอ้างโดยใช้คำสั่งต่าง ๆ

staff อ้างถึงอาเรย์ของโครงสร้าง

staff[i] อ้างถึงสมาชิกที่ i ในอาเรย์

staff[i].forename อ้างถึงชื่อหน้าของสมาชิกที่ I ของอาเรย์

staff[i].surname[j] อ้างถึงตัวอักษรตัวที่ j ในนามสกุล ของสมาชิกที่ i ของอาเรย์

การเรียกใช้งานสมาชิกบางตัวในอาร์เรย์ของโครงสร้างผ่านฟังก์ชัน

การใช้ข้อมูลสมาชิกแต่ละตัวจะอ้างถึงโดยการอ้างผ่านระบบดัชนีเหมือนอาร์เรย์ทั่วไป เช่น ฟังก์ชันที่ใช้ในการพิมพ์ชื่อสมาชิกคนที่ระบุ จะเรียกใช้โดย

```
print_person ( staff[k] );
```

รูปแบบฟังก์ชันสามารถกำหนดด้วย

```
void  print_person ( Person  employee )
```

การเรียกใช้งานสมาชิกทุกตัวในอาร์เรย์ของโครงสร้างผ่านฟังก์ชัน

หากต้องการเรียกใช้งานฟังก์ชันที่ทำงานกับทั้งอาร์เรย์ เช่น การเรียกใช้งานฟังก์ชันที่ทำการเรียงลำดับอาร์เรย์ตามชื่อหน้า จะต้องส่งอาร์เรย์และขนาดของอาร์เรย์ไปยังฟังก์ชันนั้น เช่น

```
sort_forename ( staff, STAFFSIZE );
```

รูปแบบฟังก์ชันสามารถกำหนดด้วย

```
void  sort_forename ( Person  staff[ ], int  size )
```

การกำหนดค่าเริ่มต้นให้กับอาร์เรย์ของโครงสร้าง

การกำหนดค่าเริ่มต้นให้กับอาร์เรย์ของโครงสร้างสามารถทำได้โดย

```
Person   staff[ ] = {  { “Bloggs”, “Joe”,     MALE,       21 },
                       { “Smith”,  “John”,   MALE,       30 },
                       { “Black”,  “Mary”,   FEMALE,  25 }  };
```

6.11 อาร์เรย์แบบหลายมิติ (Multi-dimensional Arrays)

จากพื้นฐานที่ผ่านมาเรื่องอาร์เรย์จะเป็นลักษณะของอาร์เรย์มิติเดียว แต่อาร์เรย์อาจจะมีมากกว่า 1 มิติก็ได้ เช่น ข้อมูลคะแนนสอบของนักศึกษาแต่ละคนภายในชั้นซึ่งแบ่งเป็นคะแนนเก็บหลายส่วน จะพบว่าหากต้องการเก็บข้อมูลคะแนนสอบของนักศึกษาแต่ละคนสามารถใช้อาร์เรย์มิติเดียว ดังตัวอย่าง

```
#define  NUMBER_OF_PAPERS   5
int  student [ NUMBER_OF_PAPERS ];
/* int  student[5];    */
```

student[0]	student[1]	student[2]	student[3]	student[4]
5.6	8.5	12.6	24.1	16.0

แต่หากเพิ่มเติมว่าให้เก็บข้อมูลคะแนนสอบของ
นักศึกษาทุกคน จะต้องใช้อาร์เรย์หลายมิติเข้ามาเกี่ยวข้อง ตัวอย่างเช่นการเก็บข้อมูลคะแนนสอบของนักศึกษา 2 คนโดยมีคะแนนสอบของการสอบทั้งสิ้น 5 ครั้ง

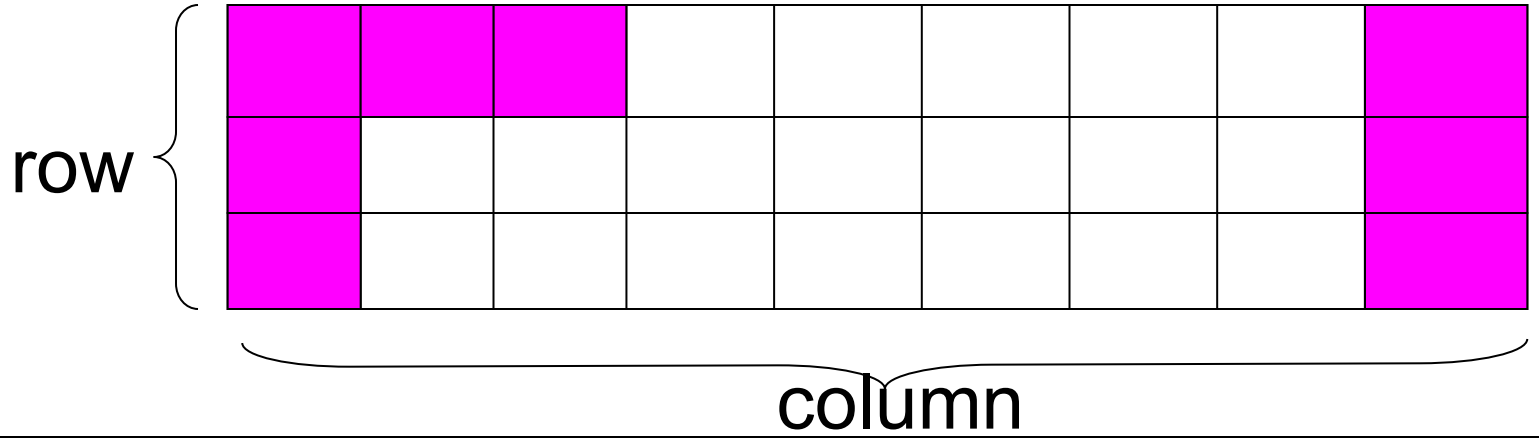
ครั้งที่	1	2	3	4	5
นาย ก	5.6	8.5	12.6	24.1	16.0
นาย ข	6.0	7.2	15.0	25.0	18.0

รูปที่ 6.10 แสดงตัวอย่างการเก็บข้อมูลคะแนนของ นักศึกษา

การอ้างอิงถึงข้อมูลในอาร์เรย์ 2 มิติ

เราจะมองอาร์เรย์ 2 มิติในลักษณะที่ประกอบด้วย แถว(row) และคอลัมน์(column) โดยข้อมูลที่อ้างอิงตัวแรกหมายถึง แถว และข้อมูลถัดมาคือ คอลัมน์

marks[2][8]



จากลักษณะความต้องการเก็บข้อมูลดังกล่าวจะต้องเตรียมอาร์เรย์เพื่อเก็บข้อมูลในลักษณะ 2 มิติ สามารถประกาศอาร์เรย์ดังนี้

```
#define  NUMBER_OF_PAPERS        5
#define  NUMBER_OF_STUDENTS      50
int
marks[NUMBER_OF_STUDENTS][NUMBER_OF_PAPERS];
/*   int marks[50][5];    */
```

โปรแกรมการรับค่าอาร์เรย์ 2 มิติ

```
#include<stdio.h>
main()
{   float  score[10][3];
    int i,j;
    printf("Please put score\n");
    for(i=0;i<10;i++)
      for(j=0;j<3;j++)
            scanf("%f",&score[i][j]);
}
```

score[0][0]	score[0][1]	score[0][2]
score[1][0]	score[1][1]	score[1][2]
score[9][0]	score[9][1]	score[9][2]

ลักษณะข้อมูล หน่วยความจำ

score[0][0]	score[0][1]	score[0][2]
score[1][0]	score[1][1]	score[1][2]
score[2][0]	score[2][1]	score[2][2]
score[9][0]	score[9][1]	score[9][2]

200	202	204	206	208	210
[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]

210	212	214	216	218	220
[2][0]	[2][1]	[2][2]	[3][0]	[3][1]	[3][2]

220	222	224	226	228	230
[4][0]	[4][1]	[4][2]	[5][0]	[5][1]	[5][2]